

منبع:

Artificial Intelligence A Modern Approach

هوش مصنوعی (رهیافتی نوین) ویراست سوم

تالیف:

استوارت راسل، پیترو نورویگ



١. مقدمه

علل مطالعه هوش مصنوعی:

- i. یادگیری بیشتر در مورد خودمان
- ii. استفاده و بهره برداری از سیستم‌هایی که توسط علم هوش مصنوعی تولید می شوند.

سیستم‌های هوشمند:

- i. **نرم افزاری:** مثل نرم افزارهای هوشمند نظیر نرم افزارهای تشخیص چهره یا تشخیص صدا
- ii. **سخت افزاری:** مثل انواع ربات ها: فوتبالیست، امدادگر، جاروبرقی و

• هوش مصنوعی چیست؟

افراد مختلف، دیدگاه‌های متفاوتی نسبت به هوش مصنوعی دارند

❖ آیا بیشتر به رفتار اهمیت می‌دهید یا به تفکر و استدلال؟

❖ آیا در ساخت سیستم‌های هوشمند، انسان را الگو قرار می‌دهید یا بر اساس یک استاندارد ایده آل هوشمندی (منطقی بودن، عقلانیت) عمل می‌کنید؟

انسان گونه فکر کردن	منطقی فکر کردن
انسان گونه عمل کردن	منطقی عمل کردن

تفاوت انسانی بودن و منطقی بودن (عقلانیت) :

منطقی بودن یعنی یک سیستم بر اساس دانش خود بهترین کار ممکن را در یک لحظه انجام دهد در حالیکه انسان در هر لحظه نمی تواند بهترین کار ممکن را در یک لحظه انجام دهد چون انسان ها کامل نیستند.

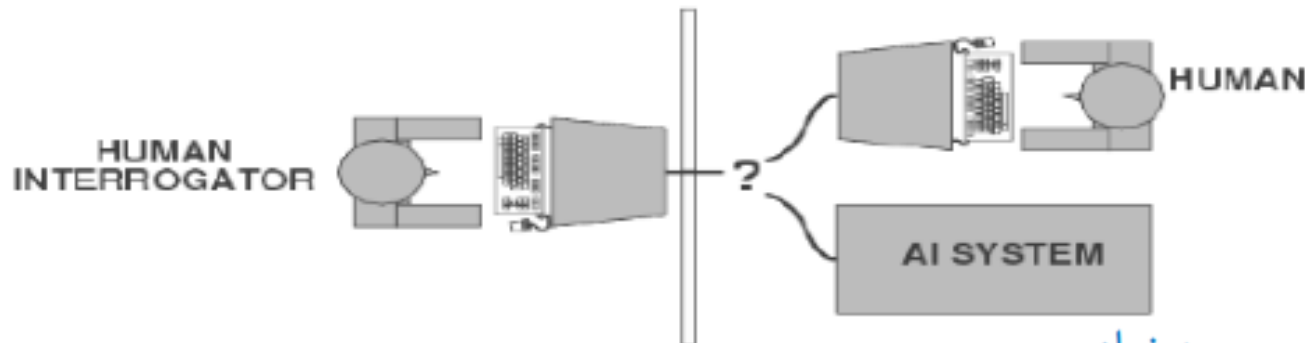
منطقی فکر کردن	مانند انسان فکر کردن
منطقی عمل کردن	مانند انسان عمل کردن

❖ نگرش مبتنی بر انسانی، جزء علوم تجربی است که شامل فرضیات و تاثیر آن توسط تجربیات است.

❖ نگرش منطقی، ترکیبی از ریاضیات و مهندسی است که با منطق سر و کار دارد.

مانند انسان عمل کردن (آزمون تورینگ):

یعنی برای تست سیستم هوشمند، به جای پیشنهاد لیستی از پارامترها و سوالات که شاید هم بحث برانگیز باشد تست تورینگ را مطرح کرد.



قابلیت‌های مورد نیاز.

▣ پردازش زبان طبیعی

▣ ذخیره و بازنمایی دانش

▣ استدلال خودکار

▣ یادگیری

❖ **بینایی کامپیوتر**^۲: برای درک اشیاء

❖ **رباتیک**^۴: برای جابجایی و کنترل اشیاء

جهت توانایی در برقراری ارتباط موفقیت آمیز :

به منظور ذخیره سازی آنچه که از قبل می‌داند و یا در حین آزمون آن را به دست می‌آورد.

جهت استفاده از اطلاعات ذخیره شده در پاسخگویی به سوالات و کسب نتایج جدید

تا خود را با شرایط تازه وفق دهد و الگوها را کشف و برون ریزی کند.

مانند انسان فکر کردن^۵ (مدل سازی شناختی):

اگر بخواهیم ادعا کنیم برنامه هوشمند ما مانند انسان فکر می کند ابتدا باید بررسی کنیم انسان چگونه فکر می کند

از طریق درون گرایی (سعی در به دست آوردن طرز تفکر):

اگر قادر به ایجاد تئوری دقیقی از ذهن باشیم آن گاه می توان این تئوری را به برنامه کامپیوتری تبدیل کرد.

از طریق آزمایشات روان شناسی:

۱. درون گرایی (به افکار خود پی ببریم)

۲. آزمایشات یا تجربیات روان شناسی (در حین فعالیت)

۳. تصویربرداری از مغز انسان (در حین فعالیت)

• درک چگونگی تفکر انسانی و عملکرد مغز

– درون گرایی

– تجارب روانشناسی

• به دنبال ایجاد تئوری دقیقی درباره عملکرد ذهن انسان و تبدیل

آن به برنامه کامپیوتری

منطقی فکر کردن^۸ (رهیافت قوانین تفکر):

- ارسطو: «فرآیند استدلال/تفکر درست چیست؟»
- مثال: "سقراط انسان است، تمام انسانها فانی هستند، پس سقراط فانی است."
- پایه ریزی منطق (Logic)
- برنامه هایی براساس قوانین تفکر برای ایجاد سیستمهای هوشمند
- موانع اصلی
 - دریافت دانش غیررسمی و تبدیل آن به دانش رسمی : «اکثر انسانها پرتلاش هستند.»
 - تفاوت میان قادر به حل مسأله بودن در تئوری و در عمل (بن بست محاسباتی)
یعنی مسائلی وجود دارند که با تعداد کمی فرضیات می تواند کامپیوتر را به بن بست محاسباتی بکشاند.

منطقی عمل کردن^۱ (رهیافت عامل منطقی):

- رفتار **منطقی**: انجام عمل درست
- عمل درست: عملی که با توجه به اطلاعات موجود، انتظار می رود شانس رسیدن به هدف را به حداکثر برساند .
- لزوماً شامل تفکر نمی باشد - مانند پلک زدن - اما تفکر باید در خدمت عمل منطقی باشد.
- یکی از راههای رسیدن به رفتار منطقی استدلال منطقی است.
- تنها راه رسیدن به عقلایی بودن استنباط صحیح نیست
- گاهی موقعیتی پیش می آید که هیچ کار درست قابل اثباتی وجود ندارد
- بعضی رفتارهای منطقی بدون استدلال منطقی (آنی) پدید می آیند

نکته: در محیط‌های پیچیده رسیدن به عقلانیت کامل (رسیدن به منطق کامل و انجام اعمال صحیح) امکان پذیر نیست چون محاسبات زیادی را می‌برد بنابراین فرضیه سودمندی را می‌پذیریم و عقلانیت کامل حالت تئوریک و تحلیلی دارد عقلانیت محدود یعنی درست عمل کردن هنگامی که وقت کافی برای انجام محاسبات وجود داشته باشد.

عامل منطقی

- **عامل:** هر چیزی که قادر به درک نمودن و عمل کردن باشد.
 - به طور انتزاعی، عامل یک تابع از تاریخچه ادراکی بر روی اعمال می باشد:
$$[f: P^* \rightarrow A]$$
 - هشدار: محدودیت های محاسباتی باعث شده اند که منطقی بودن به طور کامل، غیر قابل دسترس باشد.
- طراحی عامل منطقی برای به حداکثر رساندن سودمندی مورد انتظار.

تصمیم‌گیری

بازی

□ کاربردهای عملی بسیار

- زمانبندی، مانند زمانبندی خطوط هوایی
- مسیریابی، مانند نقشه‌های گوگل
- تشخیص پزشکی
- موتورهای جستجوی وب
- تشخیص هرزنامه
- تشخیص کلاهبرداری
- سیستم‌های توصیه‌گر
- و بسیاری از کاربردهای دیگر ...

رباتیک:

□ تشخیص اشیا و چهره

□ بخش‌بندی تصویر

□ دسته‌بندی تصاویر

- اولین پیروزی در برابر قهرمان دنیای انسانی
- بازی «هوشمند و خلاق»
- بررسی ۲۰۰ میلیون وضعیت در ثانیه
- درک ۹۹/۹ حرکت‌ها به وسیله انسان

پردازش زبان طبیعی

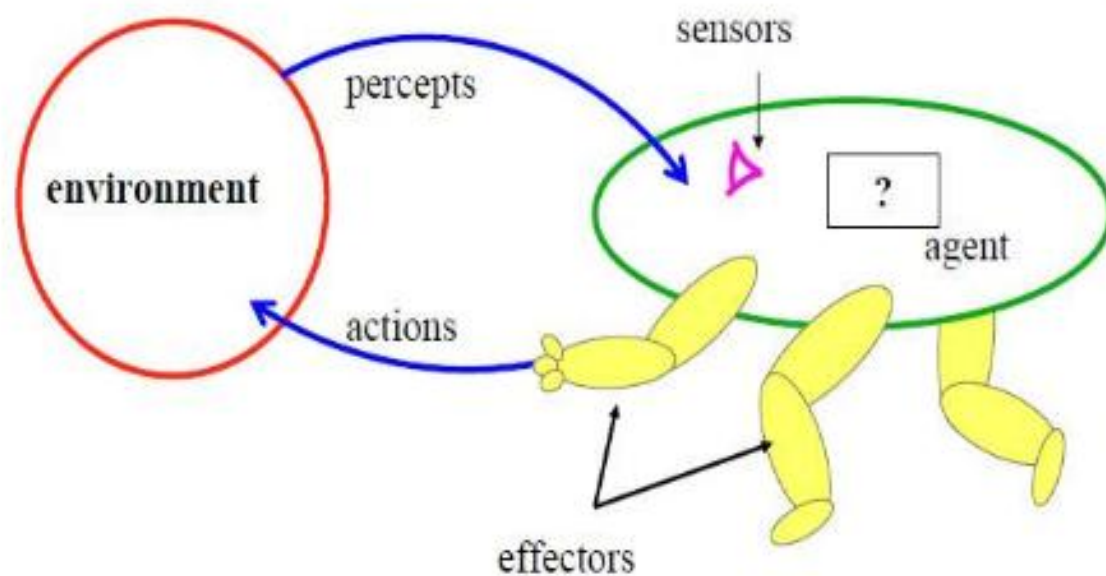
بینایی ماشین



۲. عامل های هوشمند

عامل :

هر چیزی است که محیط اطراف خود را از طریق حسگرها^۲ ادراک می کند و از طریق محرکها^۳ بر روی آن محیط عمل انجام می دهد.



مثال‌هایی از عامل‌ها:

عامل انسانی

❖ **Sensor:** گوش، چشم، پوست، زبان، بینی،

❖ **effector:** دست، پا، دهان، اندام‌های دیگر

عامل رباتیک

❖ **Sensor:** دوربین، یابنده‌های مادون قرمز

❖ **effector:** موتور، چرخ‌ها، بازوها

عامل نرم افزاری:

❖ **sensor:** صفحه کلید

❖ **effector:** صفحه نمایش

ادراک :

ورودی‌های ادراکی عامل در هر لحظه (هر آنچه که عامل از طریق سنسورهایش دریافت می‌کند.)



تابع عامل:

رفتار یک عامل توسط یک تابع عامل که هر رشته ادراکات ممکن را به یک عمل نگاشت می‌کند، توصیف می‌شود
می‌توان به صورت جدولی درآورد که عامل را توصیف می‌نماید.

مثال: دنیای جاروبرقی

محیط عامل: شامل دو فضای A,B می‌باشد که هر کدام می‌توانند تمیز یا کثیف باشند.

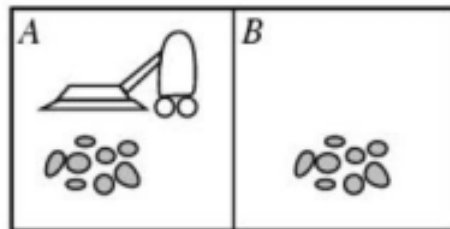
حسگرها: حسگر وضعیت، حسگر تعیین محل

ادراکات: [A,Dirty][A,Clean][B,Clean][B,Dirty]

محرک‌ها: چرخ‌ها، ابزارهای مکش

اعمال: حرکت به چپ، راست، مکش، هیچ کار

نکته: در این مثال ساده، ما می‌توانیم تمام رشته ادراکات ممکن و عملکردهای وابسته را لیست نماییم. این عامل، یک عامل مبتنی بر جدول نام دارد.



Percept sequence	Action
[A,Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
[A, Clean],[A, Clean]	Right
[A, Clean],[A, Dirty]	Suck

عامل‌ها چگونه عمل می‌کنند؟

یک عامل منطقی، عاملی است که کار درست انجام می‌دهد. کار درست، عملی است که باعث موفق تر شدن عامل می‌شود. برای اندازه گیری میزان موفقیت یک عامل از واژه‌ای به نام معیار کارایی^۱ استفاده می‌کنیم. بدیهی است که معیار کارایی یکسانی برای همه عامل‌ها وجود ندارد. معیار کارایی باید از دید **طراح عامل** تعیین شود. به عنوان مثال برای جاروبرقی معیارهای کارایی عبارتند از: میزان انرژی الکتریکی مصرف شده، میزان زباله جمع شده در چند ساعت توسط جارو برقی و میزان سر و صدای تولید شده و ...

نکته: زمان ارزیابی معیار کارایی اهمیت دارد. باید معیار کارایی را در دراز مدت ارزیابی کرد.

تفاوت میان منطقی بودن^۱ و عقل کل بودن^۲ (همه چیز دانی)

باید بین منطقی بودن و عقل کل بودن تفاوت قائل شویم:

❖ یک عامل عقل کل نتیجه واقعی اقداماتش را می‌داند و بر این اساس عمل می‌کند ولی عقل کل بودن در

واقعیت غیر ممکن است (داستان شخص پیاده رو و هواپیما)

❖ منطقی بودن، کارایی مورد انتظار را بیشینه می‌کند در حالیکه عقل کل بودن کارایی واقعی را بیشینه می‌کند و

اگر ما انتظار داشته باشیم که یک عامل آنچه را که در عمل بهترین خواهد بود انجام دهد طراحی چنین عاملی

غیر ممکن است.

نتیجه اینکه عقل کل بودن نیاز به یک دانش بی نهایت دارد، ولی منطقی بودن یعنی در حدی که به عامل، دانش

تزیق کرده‌ایم از او انتظار داشته باشیم.

تعریف عامل منطقی ایده آل:

« یک عامل منطقی ایده‌آل، به ازای هر رشته ادراکات ممکن باید اعمالی را بر اساس رشته ادراکی و دانش پیش زمینه‌ای که دارد، انجام دهد تا معیار کارایی‌اش را ماکزیمم کند. »

خودمختاری :

رفتار یک عامل می‌تواند متکی بر پایه تجربه خود و دانش درونی بنا نهاده شود. اگر عامل فقط بر اساس دانش درونی (پیش زمینه) عمل کند و به ادراکات دریافت شده از محیط توجه نکند فاقد خودمختاری است. بنابراین بهتر این است که عامل خودمختار باشد، یعنی تجربیاتش را نیز در نظر بگیرد.

در عمل به ندرت از ابتدا نیاز به خودمختاری کامل است. وقتی عامل تجربه‌ای ندارد و یا تجربه کمی دارد به صورت تصادفی عمل می‌کند، چون از ابتدا هیچ ادراکی از محیط نگرفته، بنابراین همان گونه که خداوند برای تکامل حیوانات، به اندازه کافی واکنش‌های ذاتی قرار داده است تا بتوانند آنقدر زنده بمانند و سپس خودشان یاد بگیرند، به طریق مشابه، معقول خواهد بود که برای یک عامل هوش مصنوعی مقداری دانش اولیه برای عامل فراهم کنیم. پس از کسب تجربه کافی از محیط اطرافش، رفتار یک عامل عقلانی می‌تواند به طور موثر، مستقل از دانش قبلی‌اش شود. بنابراین افزودن یادگیری، طراحی یک عامل عقلانی ساده را امکان پذیر می‌کند تا بتواند در محیط‌های گوناگون موفق باشد.

□ عامل خودمختار.

- عاملی که رفتارش به میزان تجربه‌اش از محیط بستگی داشته باشد. (نه فقط به گفته‌های طراح)
- هر چه تجربه بیشتری کسب کند، رفتارش بهتر می‌شود.
- دارای قابلیت یادگیری و تطبیق‌پذیری

تعیین مشخصات محیط کار (PEAS):

برای یک عامل، اولین مرحله تعیین مشخصات محیط کار تا حد امکان به صورت کامل می باشد. مواردی مانند مقیاس کارایی (Performance measure)، محیط (Environment)، اقدام گر ها (Actuator) و حسگر ها (Sensor) تحت عنوان محیط کار (PEAS) توصیف می شوند.



□ مثال. طراحی یک سیستم خودکار هدایت تاکسی.

- معیار کارایی: سودآوری، امنیت، سرعت و ...
- محیط: خیابان ها، مسافر ها، افراد پیاده، ماشین ها، چراغ راهنمایی و ...
- حسگر ها: دوربین ها، حسگر های صوتی، سرعت سنج، شتاب سنج، جی پی اس، کیلومتر شمار، حسگر های وضعیت موتور، صفحه کلید، میکروفون و ...
- اثرکننده ها: فرمان، شتاب دهنده، ترمزها، بوق، چراغ ها و ...

• عامل: سیستم تشخیص پزشکی

– معیار کارآیی: سلامتی بیمار، به حداقل رساندن هزینه و ...

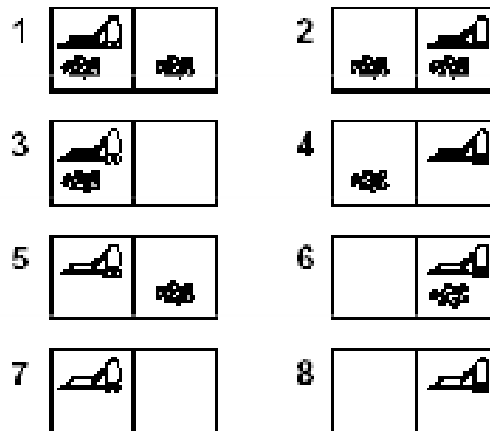
– محیط: بیمار، بیمارستان، کارمندان و ...

– اثر کننده ها: صفحه نمایش (پرسش ها، آزمایش ها، تشخیص ها،
مداوا)

– حسگرها: صفحه کلید (دریافت علائم، یافته ها و پاسخ های بیمار)

- هر محیط دارای مجموعه ای از حالت ها می باشد:
 – محیط در هر لحظه فقط در یکی از این حالت ها می باشد.

- مثال: دنیای مکش



$$S = \{1, 2, 3, 4, 5, 6, 7, 8\}$$

- در لحظه شروع، محیط در یکی از حالت های ممکن می باشد

– عمل عامل در محیط، باعث **تغییر حالت** محیط می شود



- حالت فعلی: S_i

- عمل عامل: $Action$

- حالت بعدی: S_j

- مثال: دنیای مکش



I. کاملاً رویت پذیر^۱ در مقابل نیمه رویت پذیر^۲:

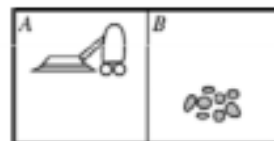
اگر حسگرهای یک عامل امکان دسترسی به وضعیت کامل محیط در هر لحظه از زمان را به عامل بدهند می‌گوییم محیط کاملاً رویت پذیر است، مانند: صفحه شطرنج، محیط پازل 8 و محیط جدول کلمات متقاطع. کار در محیط‌های کاملاً رویت پذیر آسان است، زیرا نیازی نیست که عامل، هیچ حالتی را حفظ و ذخیره کند تا بتواند اتفاقاتی که در دنیا روی می‌دهد را ثبت کند. گاهی اوقات، عدم دقت حسگرها و یا نویز باعث می‌شود قسمتهایی از حالت‌ها در داده‌های حسگر حذف شوند، اینگونه محیط‌ها، نیمه رویت پذیر هستند. محیط رانندگی تاکسی یک محیط نیمه رویت پذیر است.

- **کاملاً قابل مشاهده** (در مقابل مشاهده پذیر جزئی): محیطی که در آن در هر لحظه از زمان حسگرهای عامل به آن امکان دستیابی به حالت کامل محیط را می‌دهند.

- مثال: دنیای مکش – حسگرها: [location, status]

- تشخیص مکان: چپ یا راست
- تشخیص وضعیت: تمیز یا کثیف

[LEFT, [CLEAN, DIRTY]]



II. قطعی^۲ در مقابل غیر قطعی (اتفاقی):

اگر بر اساس وضعیت فعلی و اقدامی که توسط عامل انجام می‌شود وضعیت بعدی محیط به طور کامل تعیین شود گوییم که محیط قطعی است. در غیر این صورت اتفاقی است مثلاً محیط دنیای جاروبرقی قطعی است ولی محیط رانندگی تاکسی اتفاقی است (ممکن است چراغ راهنما قرمز شود یا بنزین تمام شود). اگر یک محیط بدون توجه به اقدامات دیگر عامل‌ها قطعی باشد محیط را استراتژیک یا راهبردی گوییم، مانند محیط بازی شطرنج.

- یعنی کنش ما قطعی ولی کنش عامل‌های دیگر مشخص و قطعی نباشد.

• **قطعی:** (در مقابل اتفاقی): حالت بعدی محیط کاملاً بوسیله حالت فعلی و عمل انجام شده توسط عامل قابل تعیین می‌باشد.

– اگر محیط به جز در مورد عمل عامل‌های دیگر قطعی باشد، آنگاه محیط **استراتژیک** می‌باشد.



III. مرحله‌ای^۴ در مقابل ترتیبی:

مرحله یا Episode شامل ادراک توسط عامل و آنگاه انجام یک عمل می‌باشد. در محیط‌های مرحله‌ای انتخاب عمل در هر مرحله تنها به خود آن مرحله بستگی دارد و در مراحل بعدی تاثیری ندارد. بسیاری از کارهای دسته بندی از این نوع هستند، مثل عاملی که قطعات معیوب را روی خط مونتاژ شناسایی می‌کند که بدون توجه به تصمیمات قبلی قطعه معیوب را شناسایی می‌کند. در محیط ترتیبی تصمیم فعلی می‌تواند بر تصمیمات بعدی تاثیر بگذارد، مانند شطرنج و رانندگی تاکسی.

نکته: محیط‌های مرحله‌ای نسبت به ترتیبی ساده تر هستند چون عامل نیاز ندارد به جلوتر فکر کند.

- **اپیزودیک** (در مقابل ترتیبی): تجربه عامل به «دوره های» غیرقابل تجزیه تقسیم می شود (هر دوره شامل ادراک عامل و سپس انجام یک عمل می باشد) و انتخاب عمل در هر دوره تنها به خود همان دوره بستگی دارد.

- مثال: روبات کنترل کننده کیفیت

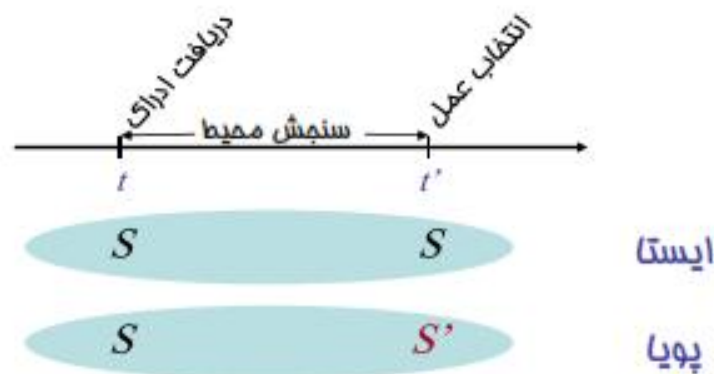


IV. ایستا در مقابل پویا:

اگر در فاصله زمانی که عامل تعمق می‌کند (یعنی از لحظه دریافت ادراک تا لحظه انتخاب عمل) محیط نیز تغییر کند می‌گوییم محیط برای آن عامل پویا است، در غیر این صورت محیط ایستاست. هوشناسی و رانندگی تاکسی محیط‌های پویا هستند. در هوشناسی چون در حین تجزیه و تحلیل، دما تغییر می‌کند و در رانندگی ممکن است چراغ‌های راهنمایی در حین رانندگی تغییر کنند. محیط جدول کلمات متقاطع و شطرنج، ایستا می‌باشد.

نکته: به محیطی که با گذشت زمان در حین سنجش شرایطش عوض نمی‌شود اما امتیاز کارایی عامل، تغییر می‌کند محیط نیمه پویا^۱ گفته می‌شود. محیط شطرنج بدون ساعت، ایستا و محیط شطرنج با ساعت نیمه پویاست.

- **ایستا** (در مقابل پویا): محیط در حین سنجش عامل (برای انتخاب عمل) تغییر نمی‌کند. اگر خود محیط با گذشت زمان تغییر نکند ولی معیار کارایی عامل تغییر کند، آنگاه محیط **نیمه پویا** می‌باشد.



V. گسسته^۲ در مقابل پیوسته:

اگر تعداد ادراکات و اعمال عامل را بتوان با اعداد گسسته بیان کرد محیط گسسته، در غیر اینصورت محیط پیوسته می-باشد. محیط بازی شطرنج و جاروبرقی یک محیط گسسته ولی رانندگی تاکسی یک محیط پیوسته است.

- **گسسته** (در مقابل پیوسته): محیطی که در آن تعداد محدود و متمایزی از درک ها و عمل های کاملاً واضح تعریف شده باشد.

- در محیط گسسته، مجموعه حالات محیط یک مجموعه گسسته می باشد و حالات بسادگی قابل تمایز می باشند.
– مثال: محیط دنیای مکش

- $State = \{1, 2, \dots, 8\}$
- $Action = \{Left, Right, Suck, NoOp\}$
- $Percept = \{[Left, Clean], [Left, Dirty], [Right, Clean], \dots\}$

VI. تک عامله در مقابل چند عامله:

محیطی تک عامله است که فقط یک عامل در آن قرار دارد مانند محیطی که یک عامل در آن جدول کلمات متقاطع را حل می‌کند یا محیط جاروبرقی. در محیط‌های چند عامله، بیش از یک عامل در محیط قرار دارد، مانند محیط بازی شطرنج، که یک محیط دو عامله است. محیط‌های چند عامله به دو دسته محیط‌های چندعامله رقابتی و چند عامله مشارکتی تقسیم می‌شوند. در محیط چندعامله رقابتی، مقیاس کارایی عامل‌ها در تناقض با یکدیگر است مانند محیط بازی شطرنج ولی در چندعامله مشارکتی، مقیاس کارایی عامل‌ها در راستای یکدیگر است مانند محیط رانندگی تاکسی با مقیاس کارایی اجتناب از تصادف. در حالیکه اگر مقیاس کارایی در این محیط پارک کردن در یک محل خاص باشد محیط چند عامله رقابتی می‌باشد.

نکته: حالات نیمه رویت پذیر، اتفاقی، ترتیبی، پویا، پیوسته و چند عامله جزء مشکل ترین محیط‌ها می‌باشند مانند محیط رانندگی تاکسی و محیط سیستم تشخیص پزشکی.

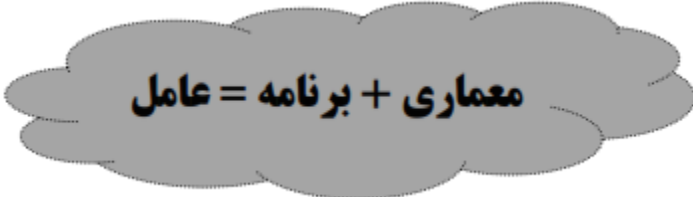
رانندگی تاکسی	شطرنج بدون ساعت	شطرنج با ساعت	
خیر	بله	بله	کاملاً قابل مشاهده
خیر	استراتژیک	استراتژیک	قطعی
خیر	خیر	خیر	دوره ای
خیر	بله	نیمه پویا	ایستا
خیر	بله	بله	گستره
خیر	خیر	خیر	تک عاملی

ساختار عامل‌های هوشمند

وظیفه طراح هوش مصنوعی، طراحی برنامه عامل است یعنی طراحی تابعی که دنباله ادراکی را به یک عمل نگاشت می‌کند.

به طور کلی معماری، ابتدا مشاهدات و ادراکات را از حسگرها می‌گیرد و برای برنامه عامل قابل دسترس می‌کند، سپس برنامه عامل را اجرا نموده و اعمال انتخاب شده را به محرک‌ها می‌رساند. ارتباط بین عامل، معماری و برنامه به صورت زیر می‌باشد.

برنامه عامل نحوه انتخاب عمل را مشخص می‌کند.



معماری + برنامه = عامل

طراحی مناسب برای هر عامل به طبیعت محیط بستگی دارد.

عامل‌های مبتنی بر جدول :

این روش برای عامل‌هایی که تعداد ادراکات آن کم و معلوم باشد مناسب است.

- ❖ جدول مورد نیاز برای عامل ساده‌ای که تنها قادر به بازی شطرنج باشد 35^{100} سطر خواهد داشت.
- ❖ زمان بسیار طولانی لازم است تا طراح قادر به ساخت جدول باشد.
- ❖ عامل مبتنی بر جدول فاقد هرگونه خودمختاری است زیرا محاسبه بهترین عمل، کاملاً درونی صورت می‌گیرد و اگر چنانچه شرایط محیط به گونه‌ای غیر قابل پیش بینی تغییر کند، عامل شکست خواهد خورد.
- ❖ حتی اگر به عامل، روش یادگیری داده شود تا درجه‌ای از خودمختاری حاصل شود برای یادگیری مقدار صحیح از بین انبوه سطرهای جدول به زمان بی نهایت نیاز خواهد بود.

Let \mathcal{P} be the set of possible percepts and let T be the lifetime of the agent. The lookup table will contain $\sum_{t=1}^T |\mathcal{P}|^t$

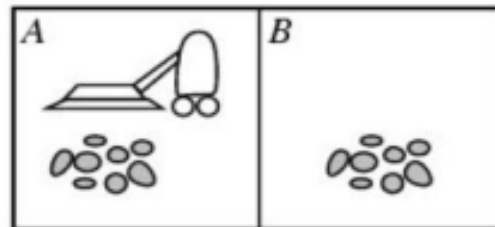
function TABLE-DRIVEN-AGENT(*percept*) **returns** an action

static: *percepts*, a sequence, initially empty
table, a table of actions, indexed by percept sequence,
initially fully specified

append *percept* to the end of *percepts*

action \leftarrow LOOKUP(*percepts*, *table*)

return *action*



Percept sequence	Action
[A, Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
[A,Clean], [A, Clean]	Right
[A,Clean], [A, Dirty]	Suck

عامل‌های واکنشی ساده:

ساده ترین نوع عامل، عامل واکنشی ساده است. این عامل‌ها اقدامات را بر اساس ادراک فعلی انتخاب می‌کنند و تاریخچه ادراکات را نادیده می‌گیرند، یعنی حافظه ندارند. برای مثال عامل جاروبرقی یک عامل واکنشی ساده است زیرا تصمیم آن تنها بر اساس محل فعلی محیط و اینکه آیا آن مکان دارای زباله است یا نه، می‌باشد. به خاطر حذف سابقه ادراک برنامه عامل در مقایسه با جدول آن بسیار کوچک است. بعنوان مثال در برنامه عامل جاروبرقی در مقایسه با جدول آن تعداد حالات ممکن از 4^T به 4 کاهش می‌یابد.

- ساده ترین نوع عامل
- در هر لحظه، عمل تنها بر اساس درک فعلی انتخاب می شود
- مثال:

```
function REFLEX-VACCUM-AGENT( [location, status]) returns an action
```

```
if status = Dirty then return Suck  
else if location = A then return Right  
else if location = B then return Left
```

- شامل قوانین شرط-عمل مانند:
– “اگر چراغ ترمرز اتوموبیل جلویی روشن شد، آنگاه ترمرز کن”

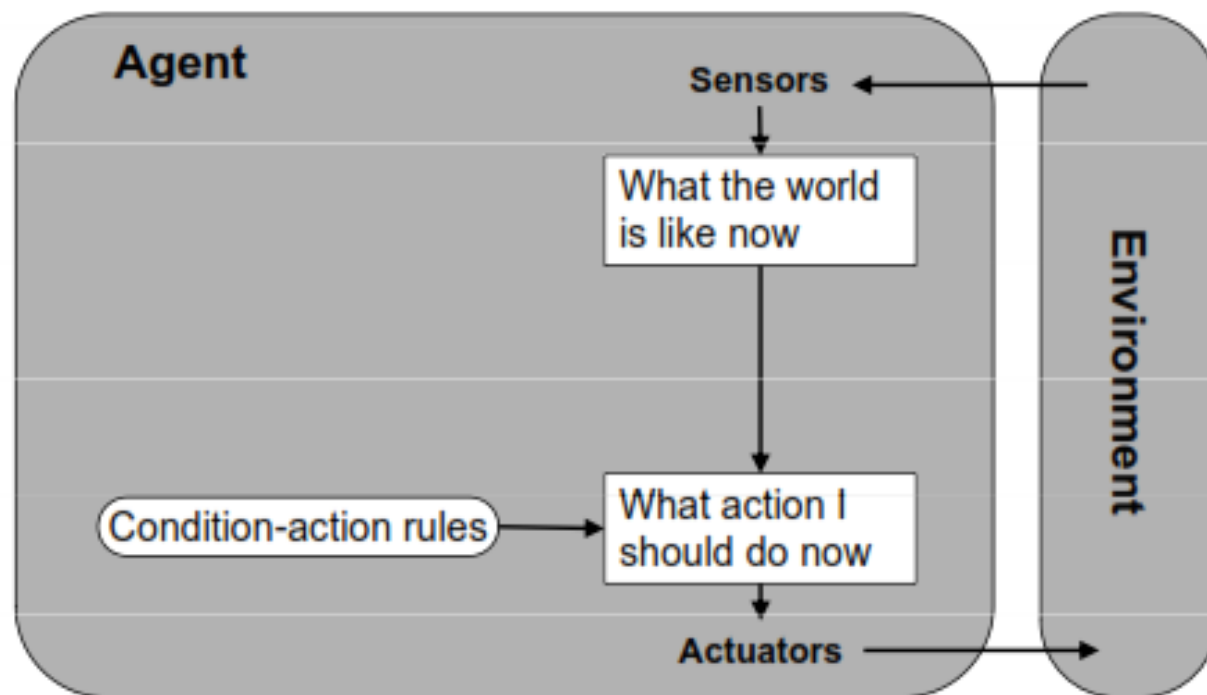
نکته: انتخاب عمل بر اساس یک سری قوانین شرط - عمل انجام می‌شوند. قوانین شرط - عمل^۱ از دو نوع اکتسابی و غریزی هستند.

❖ **اکتسابی:** اگر ماشین جلویی چراغ زد، آنگاه توقف کن.

❖ **غریزی:** اگر شی‌ای به چشم شما نزدیک شود، آنگاه چشم‌های خود را ناخودآگاه می‌بندیم.

نکته: عامل‌های واکنشی ساده این ویژگی قابل ستایش را دارند که ساده هستند ولی کاربرد و هوشمندی بسیار محدودی دارند. عامل واکنشی ساده تنها در صورتی کار می‌کند که بر اساس ادراک فعلی بتواند درست تصمیم بگیرد و این یعنی اینکه محیط کاملاً رویت پذیر باشد. حتی اگر به مقدار اندکی عدم رویت پذیری وجود داشته باشد می‌تواند مشکلات بسیار جدی بروز کند. در مورد عامل‌های واکنشی ساده که در محیط‌های نیمه رویت پذیر کار می‌کنند حلقه-های بی نهایت اغلب غیر قابل اجتناب هستند. اگر عامل بتواند اقداماتش را تصادفی کند گریز از حلقه‌های بی نهایت امکان پذیر است. بنابراین یک عامل واکنشی ساده تصادفی شده ممکن است از یک عامل واکنشی ساده قطعی بهتر عمل کند.

در محیط‌های تک عاملی، تصادفی عمل کردن، چندان منطقی نیست ولی در محیط‌های چند عامله عاقلانه است.



```
function SIMPLE-REFLEX-AGENT( percept) returns an action
  static: rules, a set of condition-action rules

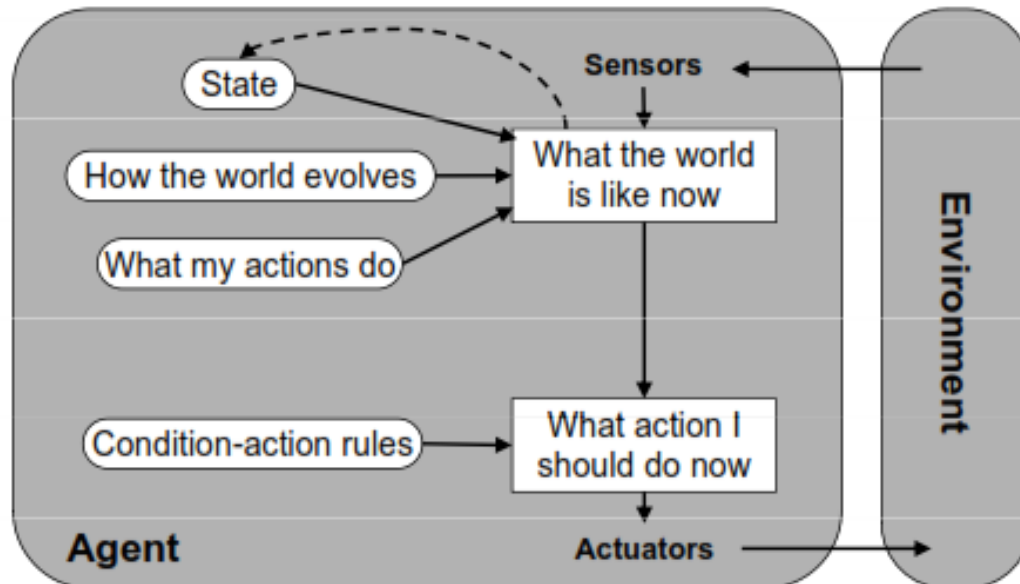
  state  $\leftarrow$  INTERPRET-INPUT( percept)
  rule  $\leftarrow$  RULE-MATCH( state, rules)
  action  $\leftarrow$  RULE-ACTION[ rule]
  return action
```

عامل واکنشی مبتنی بر مدل:

همانطور که در روش قبل (عامل واکنشی ساده) اشاره شد گاهی اوقات محیط نیمه رویت پذیر است. موثر ترین روش برخورد عامل با محیط نیمه رویت پذیر نگه داشتن سوابق آن بخش از دنیا است که اکنون عامل نمی‌تواند آن را ببیند. این بدان معنی است که عامل باید به نوعی حالات داخلی را نگه داری کند که به تاریخچه ادراکات وابسته است. بهنگام سازی اطلاعات وضعیت داخلی همزمان با گذر زمان نیازمند دو نوع دانش کد شده در برنامه عامل است. نیازمند آن هستیم که برخی اطلاعات در باره چگونگی تغییر جهان، مستقل از عامل را داشته باشیم. نیازمند اطلاعات در مورد تاثیر اعمال خود عامل بر روی محیط می باشیم.

نکته: این دانش که درباره چگونگی عملکرد دنیا می‌باشد مدل دنیا^۱ نامیده می‌شود. و عاملی که از

چنین مدلی استفاده می‌کند، **مبتنی بر مدل** نامیده می‌شود.



```

function MODEL-BASED-REFLEX-AGENT(percept) returns an action
  persistent state, the agent's current conception of the world state
               model, a description of how the next state depends on current state and action
               rules, a set of condition—action rules
               action, the most recent action, initially none

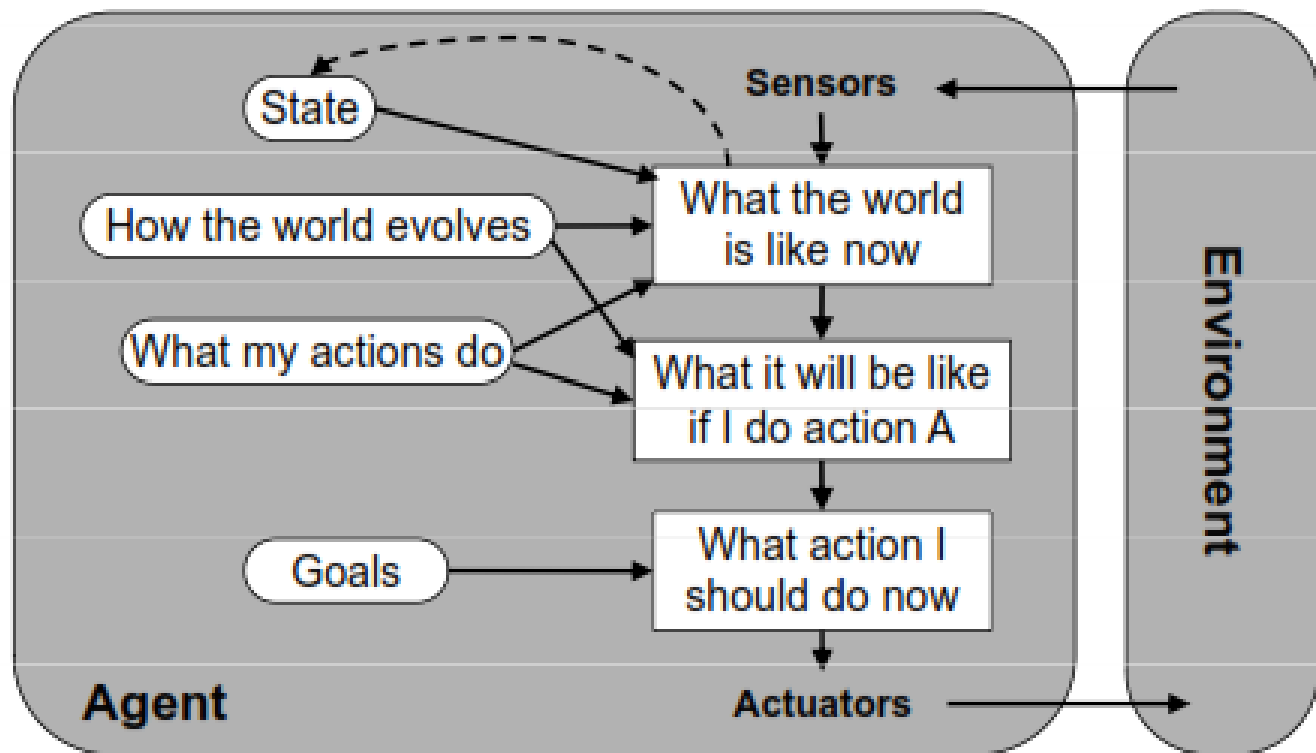
  state ← UPDATE-STATE(state, action, percept, model)
  rule ← RULE MATCH(state,
  action ← rule.ACTION
  return action

```

Figure 2.12 A model-based reflex agent. It keeps track of the current state of the world, using an internal model. It then chooses an action in the same way as the reflex agent.

عامل‌های مبتنی بر هدف:

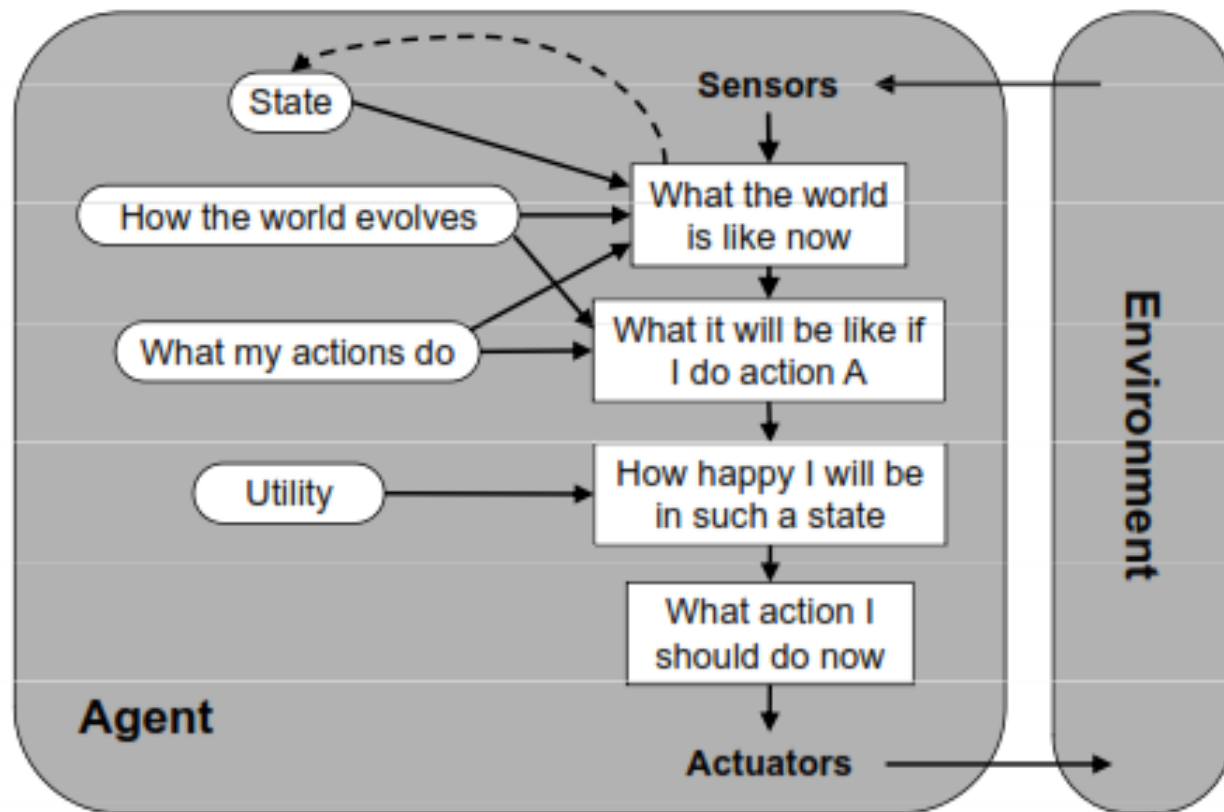
اطلاع از وضعیت فعلی محیط، همیشه برای تصمیم‌گیری در مورد اقدام بعدی کافی نیست. برای مثال در سر یک چهار راه، تاکسی می‌تواند به چپ، راست، یا مستقیم برود. تصمیم درست به مقصد مسافر بستگی دارد. بنابراین همانگونه که عامل نیازمند دانستن وضعیت جاری و قبلی است نیاز به یک هدف نیز خواهد داشت تا تصمیم‌گیری‌های وی بر مبنای آن هدف جهت گیرد. رسیدن به هدف گاهی اوقات ساده و گاهی اوقات پیچیده است. در مواقعی که هدف ساده است، ارضای هدف بلافاصله بعد از انجام یک عمل نتیجه خواهد شد ولی در مواقعی که هدف پیچیده باشد، عامل باید دنباله اقدامات طولانی را برای رسیدن به هدف طی کند. در مواقع پیچیده، جستجو^۱ و برنامه‌ریزی^۲ به یافتن دنباله‌ای از اعمال منجر خواهد شد. در مدل هدف‌گرا نمی‌توان از قوانین شرط – عمل گذشته تبعیت کرد



عوامل‌های مبتنی بر سودمندی:

اهداف به تنهایی برای ایجاد رفتاری با کیفیت و سودمندی بالا کافی نخواهد بود. برای مثال در رسیدن تاکسی به مقصد ممکن است دنباله زیادی از اعمال وجود داشته باشد تا به مقصد برسیم ولی بعضی از این مسیرها سریعتر، امن تر و ارزانتر از بقیه هستند. اهداف فقط بین حالات راضی و ناراضی تفاوت قائل می‌شوند و درباره اینکه یک حالت چقدر عامل را راضی می‌کند سخن نمی‌گویند. برای مقایسه بین حالت‌ها که میزان راضی بودن را تعیین می‌کند از تابع سودمندی استفاده می‌کنیم.

تابع سودمندی^۱: یک حالت یا رشته‌ای از حالات را به یک عدد حقیقی که درجه رضایت نام دارد، نگاشت می‌کند.





۲. جستجو (بخش ۱)

عامل‌های حل مسئله:

نوعی از عامل‌های هدف‌گرا هستند، که توسط یافتن ترتیب عملیات تصمیم می‌گیرند، چه نوع عملی را انجام دهند، تا به حالت مطلوب سوق پیدا کنند. مراحل زیر باید توسط یک عامل حل مسئله انجام شود:

فرموله کردن (تدوین) هدف: وضعیت‌های مطلوب نهایی کدامند.

فرموله کردن مسئله: چه اقدامات و وضعیت‌هایی برای رسیدن به هدف موجود است.

جستجو: در این مرحله عامل تصمیم می‌گیرد که چه رشته اعمالی می‌تواند وی را از حالت شروع به حالت هدف برساند. مسلماً این رشته از اعمال از بین یک مجموعه اعمال ممکن انتخاب می‌شود، خروجی این مرحله یک راه-حل^۱ است.

اجرا: راه‌حلی که از مرحله قبل به دست آمده اجرا می‌شود.

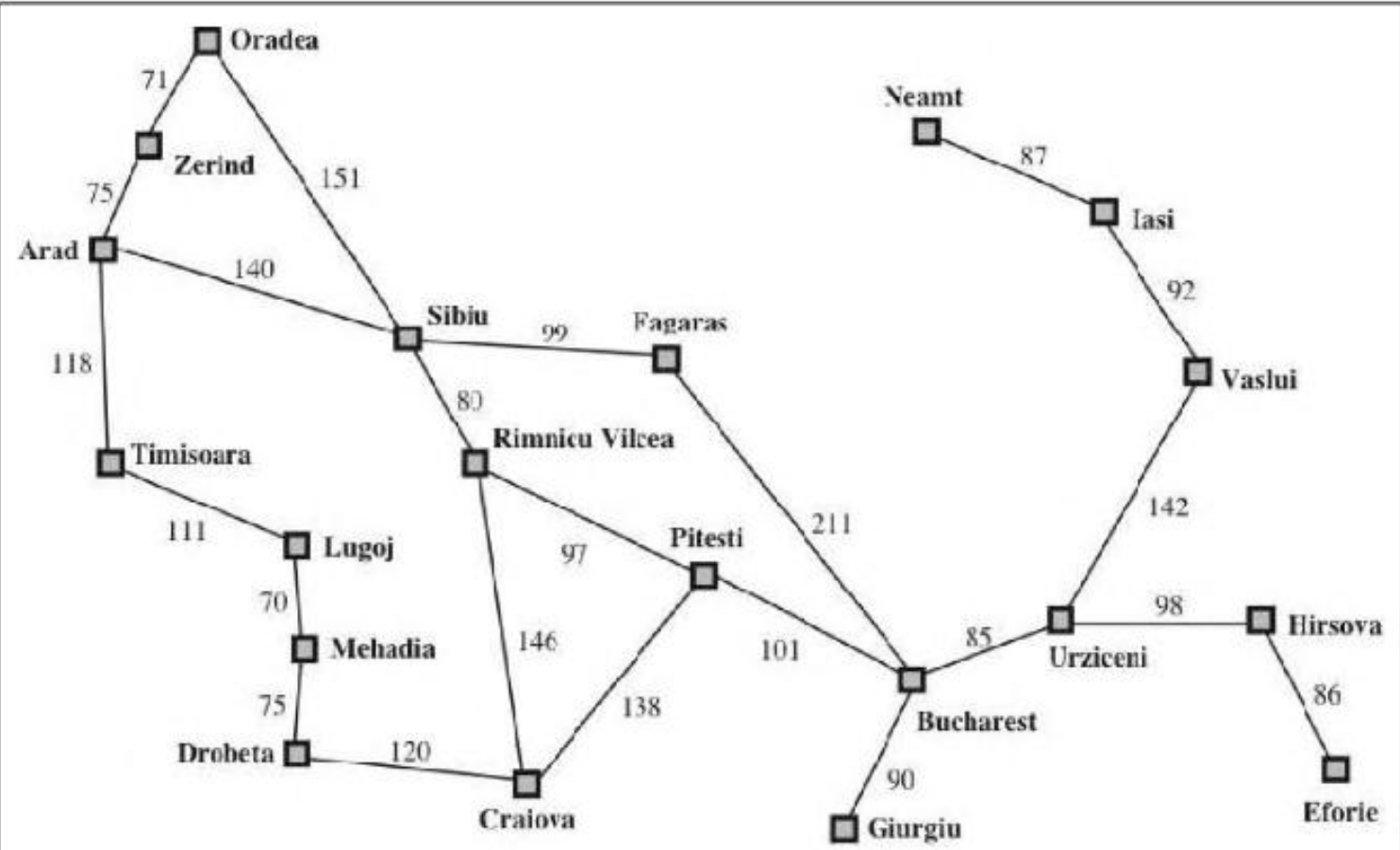


Figure 3.2 A simplified road map of part of Romania.

صورت مسئله: رفتن از آراد به بخارست

فرموله کردن هدف: رسیدن به بخارست.

فرموله کردن مسئله:

❖ **وضعیت‌ها:** شهرهای مختلف

❖ **فعالیت‌ها:** حرکت بین شهرها

جستجو: دنباله‌ای از شهرها مثل آراد، سیبیو، فاگارس، بخارست. (جستجو با توجه به کم‌هزینه‌ترین مسیر)

تعریف مسئله: مجموعه‌ای از اطلاعات است که عامل از آنها برای این که چه عملی را انجام دهد، استفاده می‌کند. یک

مسئله با موارد زیر تعریف می‌شود:

حالت اولیه: حالتی که عامل از آن شروع می‌کند. در مثال رومانی شهر آراد (Arad) n

تابع جانشین: توصیفی از حالت‌های ممکن که برای عامل مهیاست.

$$S(\text{Arad}) = \{\text{zerind} - \text{Sibiu} - \text{Timisoara}\}$$

فضای حالت: مجموعه‌ای از حالت‌ها که از حالت اولیه می‌توان به آنها رسید. در مثال رومانی: کلیه شهرهایی که با شروع از آراد می‌توان به آنها رسید.

نکته: فضای حالت = حالت اولیه + تابع جانشین.

آزمون هدف: تعیین می‌کند که آیا حالت خاصی حالت هدف است یا خیر. دو نوع هدف داریم:

❖ **هدف صریح:** در مثال رومانی رسیدن به بخارست.

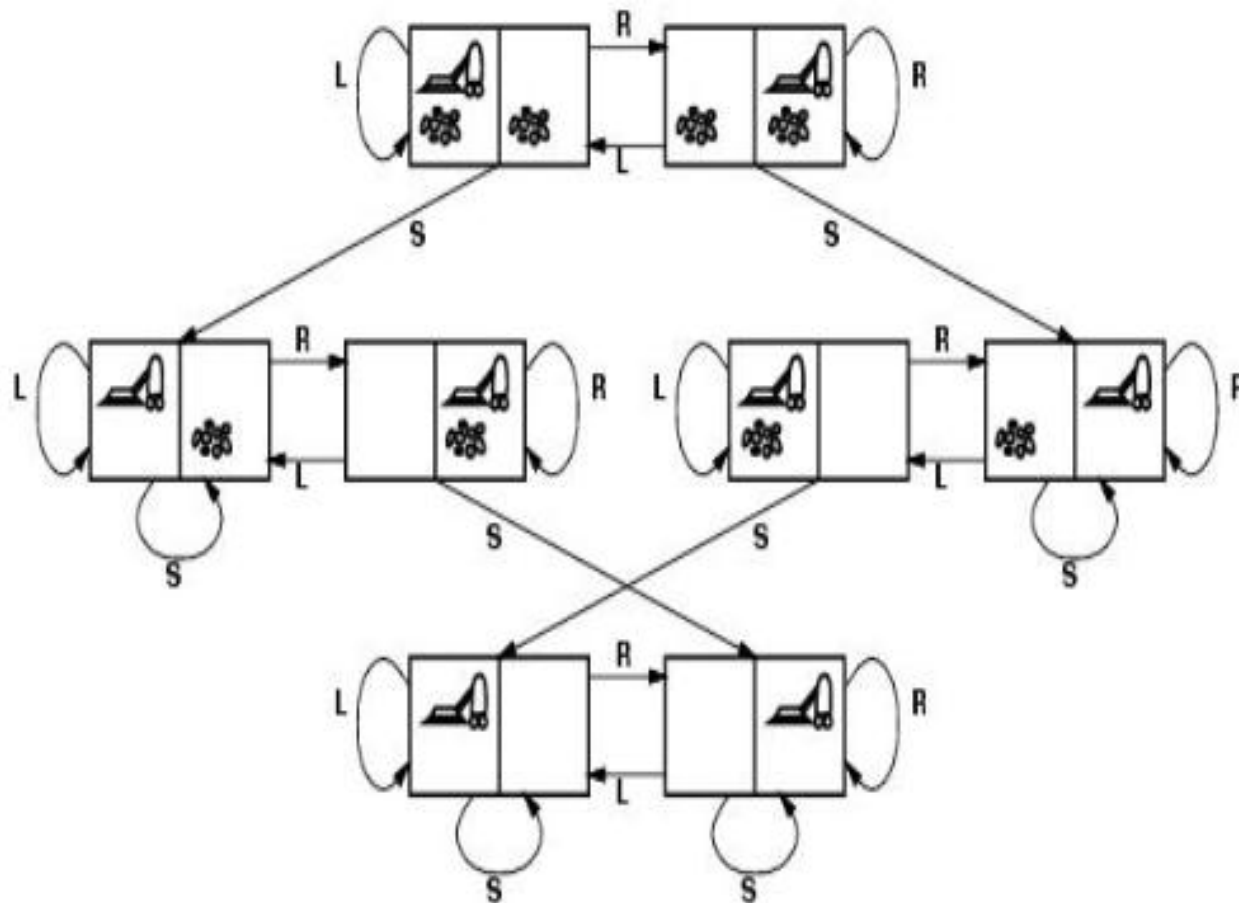
❖ **هدف انتزاعی (ضمنی):** در مثال شطرنج، رسیدن به حالت کیش و مات.

مسیر: دنباله‌ای از حالت‌ها که دنباله‌ای از فعالیت‌ها را به هم متصل می‌کند. در مثال رومانی: Arad, Sibiu, Fagaras یک مسیر است.

هزینه مسیر: برای هر مسیر یک هزینه عددی در نظر می‌گیرد. در مثال رومانی: طول مسیر بین شهرها بر حسب کیلومتر.

راه حل مسئله: مسیری از حالت اولیه به حالت هدف است. راه‌حل بهینه کمترین هزینه را دارد.

نکته: فرآیند حذف جزییات از یک توصیف، تجرید¹ یا انتزاع نام دارد. به عنوان نمونه در مثال رومانی یک سری از توصیفات در دنیای واقعی وجود دارند که به مسئله پیدا کردن یک مسیر به بخارست ربطی ندارد. مثلاً همراهان مسافر، منظره بیرون از پنجره، فاصله ایستگاه تا پمپ‌بنزین و ...



حالتها: دو مکان که هر یک ممکن است کثیف یا تمیز باشند. لذا $2 \times 2^2 = 8$ حالت در این جهان وجود دارد.

حالت اولیه: هر حالتی می‌تواند به عنوان حالت اولیه طراحی شود.

تابع جانشین: حالت‌های معتبر از سه عملیات: راست، چپ، مکش.

آزمون هدف: تمیزی تمام مربع‌ها.

هزینه مسیر: تعداد مراحل در مسیر.

نکته: اگر محیطی n محل داشته باشد $n \times 2^n$ حالت خواهد داشت.

	1	2	7	2	4
3	4	5	5		6
6	7	8	8	3	1

Goal State

Start State

حالتها: مکان هر هشت خانه شماره دار و خانه خالی در یکی از 9 خانه

حالت اولیه: هر حالتی را میتوان به عنوان حالت اولیه در نظر گرفت.

تابع جانشین: حالت‌های معتبر از چهار عمل، انتقال خانه خالی به چپ، راست، بالا یا پایین

آزمون هدف: بررسی می‌کند که حالتی که اعداد به ترتیب چیده شده‌اند (طبق شکل روبرو) رخ داده یا نه

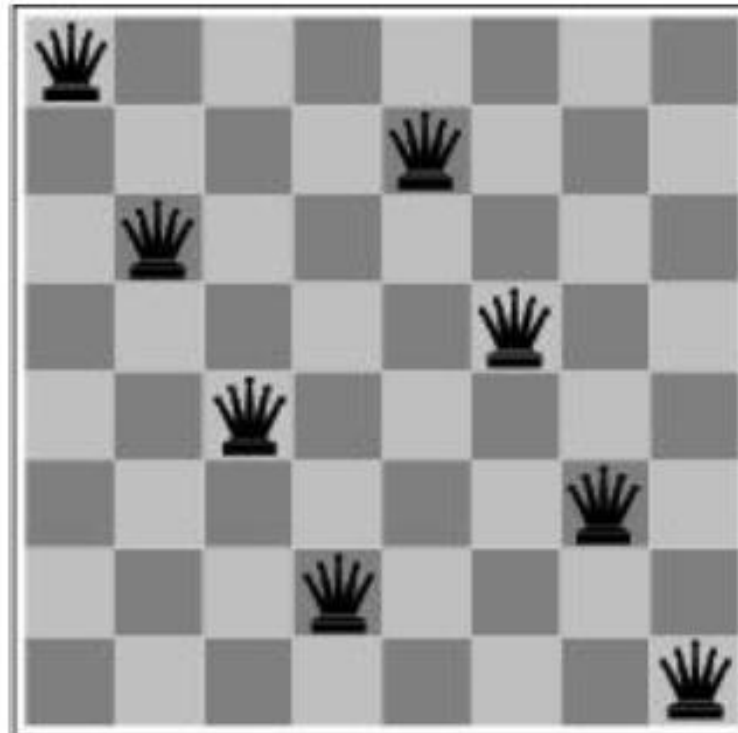
هزینه مسیر: برابر با تعداد مراحل در مسیر.

مسئله 8 وزیر:

هدف در این مسئله قرار دادن 8 وزیر روی صفحه شطرنج است به نحوی که هیچ وزیری به دیگری حمله نکند. برای این مسئله دو نوع فرموله‌سازی وجود دارد:

❖ **فرموله سازی افزایشی^۱:** که با صفحه خالی شروع شده و در هر عمل یک وزیر به صفحه اضافه می‌شود.

❖ **فرموله سازی حالت کامل^۲:** با همه 8 وزیر روی صفحه شروع می‌کند و در هر عمل یک وزیر به اطراف حرکت داده می‌شود.



فرمول بندی افزایشی

حالت‌ها: هر ترتیبی از 0 تا 8 وزیر در صفحه، یک حالت است

حالت اولیه: هیچ وزیری در صفحه نیست

تابع جانشین: وزیری را به خانه خالی اضافه می‌کند

آزمون هدف: 8 وزیر در صفحه وجود دارند و هیچ کدام به یکدیگر گارد نمی‌گیرند

در این فرمول بندی باید $10^{14} * 3$ دنباله ممکن بررسی می‌شود

فرمول بندی حالت کامل

حالت‌ها: چیدمان n وزیر ($0 \leq n \leq 8$)، بطوری که در هر ستون از n ستون سمت چپ، یک وزیر قرار گیرد و

هیچ دو وزیری بهم گارد نگیرند

حالت اولیه: با 8 وزیر در صفحه شروع می‌شود

تابع جانشین: وزیری را در سمت چپ‌ترین ستون طوری جابجا می‌کند، بطوری که هیچ وزیری آن را گارد ندهد.

آزمون هدف: 8 وزیر در صفحه وجود دارند و هیچ کدام به یکدیگر گارد نمی‌گیرند

در فرموله سازی حالت کامل، فضای حالت از $10^{14} * 3$ به 2057 کاهش می‌یابد و پیدا کردن راه‌حل در فضای

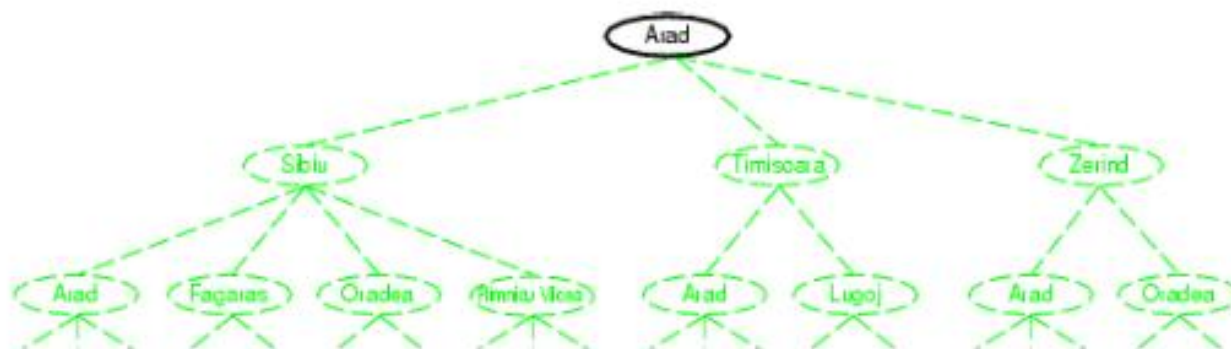
حالت کامل راحت‌تر از افزایشی است. برای فرموله کردن مسئله 100 وزیر در افزایشی 10^{400} حالت و در روش

فرموله سازی حالت کامل 10^{52} برای پیدا کردن راه حل باید بررسی شود.

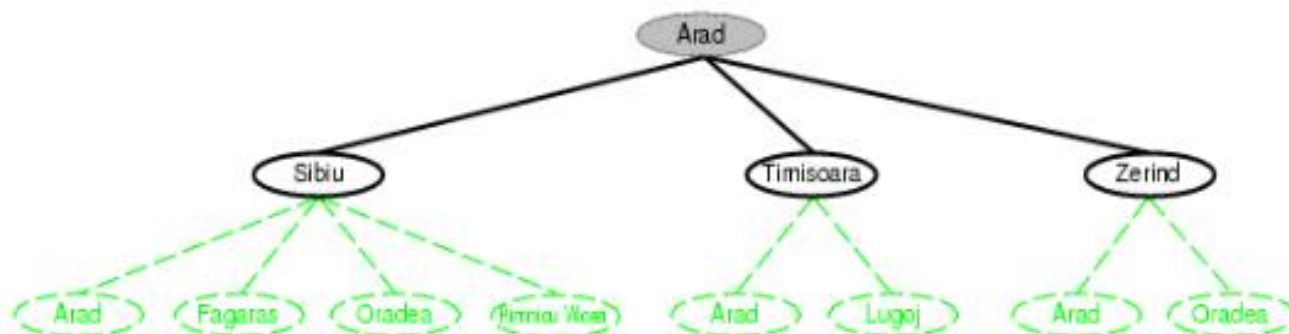
گسترش^۱:

به معنی بکارگیری تابع مابعد برای یک حالت و تولید یک مجموعه جدید از حالات می‌باشد. مجموعه نودهایی که تولید شده‌اند اما هنوز گسترش نیافته‌اند مجموعه حاشیه^۲ نامیده می‌شود و هر عنصر از مجموعه حاشیه یک نود برگ است یعنی نودی که هنوز هیچ مابعدی در درخت جستجو ندارد.

حالت اولیه



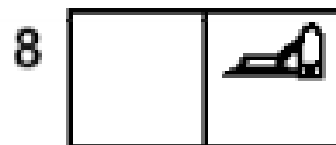
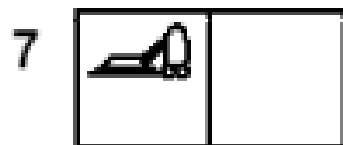
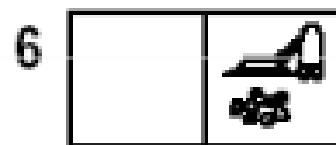
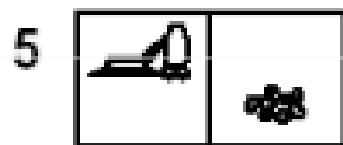
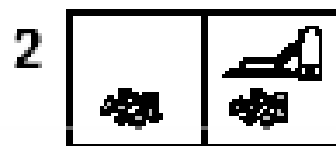
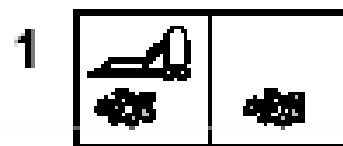
پس از گسترش آراد

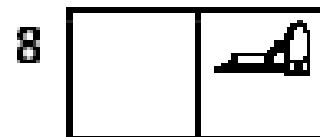
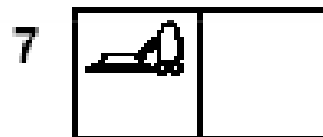
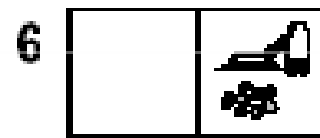
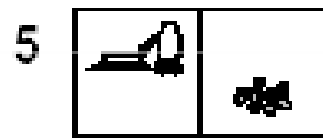
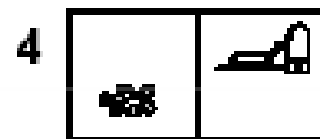
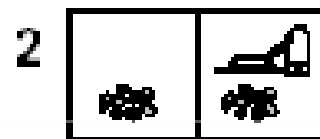


انواع مسأله

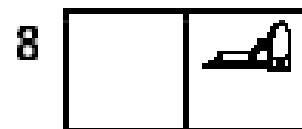
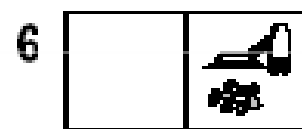
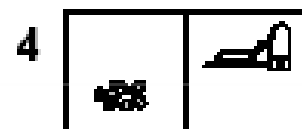
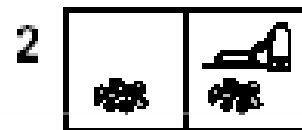
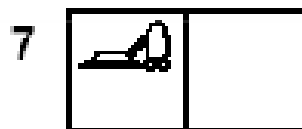
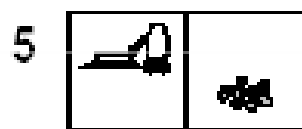
- قطعی، کاملاً مشاهده پذیر ← مسائل تک - حالت
- عامل دقیقاً می داند در چه حالتی خواهد بود؛ راه حل یک دنباله می باشد.
- قطعی، مشاهده پذیر جزئی ← مسائل چند-حالت
- ممکن است عامل ایده ای درباره اینکه کجاست نداشته باشد؛ راه حل یک دنباله است.
- غیر قطعی و/یا مشاهده پذیر جزئی ← مسائل احتمالی
- ادراک اطلاعات جدیدی درباره حالت فعلی فراهم می کند.
- در حین اجرا باید از حسگرها استفاده کند.
- راه حل به صورت یک درخت
- اغلب جستجو و اجرا به صورت یک در میان (interleave)
- فضای حالت ناشناخته ← مسائل اکتشافی (online)

• تک-حالت، شروع در #5.
راه حل؟ [Right, Suck]





- چند-حالت، شروع در
 $\{1, 2, 3, 4, 5, 6, 7, 8\}$ مثال عمل *Right*
 به $\{2, 4, 6, 8\}$.
راه حل؟
[Right, Suck, Left, Suck]



احتمالی

- غیر قطعی: مکش می تواند یک فرش تمیز را کثیف کند.

- درک محلی: گرد و خاک در محل فعلی

- ادراک: [L, Clean] یعنی شروع در #5 یا #7

راه حل؟

[Right, if dirt then Suck]

الگوریتم های جستجوی درخت

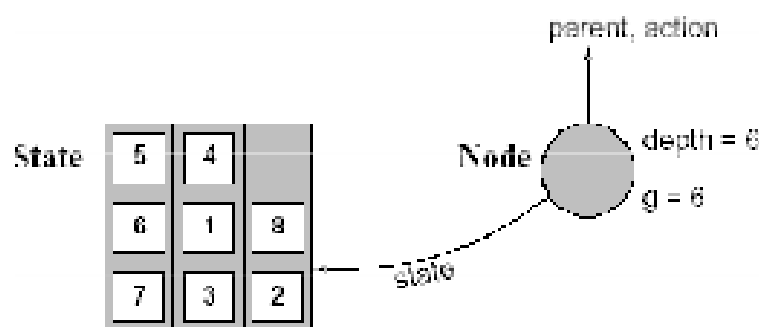
- ایده اصلی: کاوش *offline* و شبیه سازی شده فضای حالت بوسیله تولید حالات بعدی حالت هایی که تا کنون تولید شده اند.

```
function TREE-SEARCH(problem, strategy) returns a solution, or failure
  initialize the search tree using the initial state of problem
  loop do
    if there are no candidates for expansion then return failure
    choose a leaf node for expansion according to strategy
    if the node contains a goal state then return the corresponding solution
    else expand the node and add the resulting nodes to the search tree
```

```
function TREE-SEARCH(problem, fringe) returns a solution, or failure
  fringe  $\leftarrow$  INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node  $\leftarrow$  REMOVE-FRONT(fringe)
    if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
    fringe  $\leftarrow$  INSERT-ALL(EXPAND(node, problem), fringe)
```

```
function EXPAND(node, problem) returns a set of nodes
  successors  $\leftarrow$  the empty set
  for each action, result in SUCCESSOR-FN[problem](STATE[node]) do
    s  $\leftarrow$  a new NODE
    PARENT-NODE[s]  $\leftarrow$  node; ACTION[s]  $\leftarrow$  action; STATE[s]  $\leftarrow$  result
    PATH-COST[s]  $\leftarrow$  PATH-COST[node] + STEP-COST(node, action, s)
    DEPTH[s]  $\leftarrow$  DEPTH[node] + 1
    add s to successors
  return successors
```

- یک **حالت** (بیانگر) یک پیکره بندی فیزیکی می باشد
- یک **گره** یک ساختار داده ای تشکیل دهنده بخشی از درخت جستجو شامل: **پدر**، **فرزندان**، **عمق** و **هزینه مسیر** $g(X)$ است.
- **حالت ها**:



- تابع **EXPAND** گره های جدید ایجاد می کند، فیلدهای مختلف را مقدار می دهد و با استفاده از تابع **SUCCESSORS-FN** مسأله، حالت های مربوطه ایجاد می شود.

اندازه گیری کارایی حل مسئله :

خروجی الگوریتم جستجو راه حل^۱ یا شکست^۲ است. کارایی الگوریتم جستجو با چهار معیار زیر ارزیابی می شود:

❖ **کامل بودن^۳:** آیا الگوریتم تضمین می کند که در صورت وجود راه حل، راه حل را پیدا کند؟

❖ **بهینگی^۴:** آیا الگوریتم تضمین می کند که از بین چندین راه حل، راه حل بهینه یا کم هزینه ترین را پیدا کند؟

❖ **پیچیدگی زمانی^۵:** چه مدت زمانی طول می کشد تا الگوریتم جستجو، راه حل را پیدا کند؟

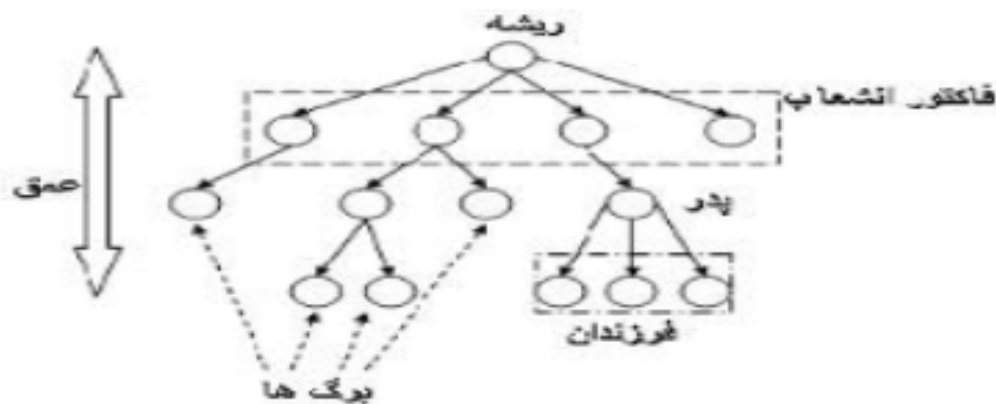
❖ **پیچیدگی فضایی^۶:** الگوریتم جستجو برای پیدا کردن راه حل، چقدر حافظه نیاز دارد؟

نکته: پیچیدگی زمانی و مکانی بر اساس سه مفهوم زیر بیان می شود:

❖ **b:** حداکثر فاکتور انشعاب درخت جستجو.

❖ **d:** عمق کم هزینه ترین راه حل.

❖ **m:** حداکثر عمق فضای حالت که می تواند بی نهایت نیز باشد.



نکته: معمولاً پیچیدگی زمانی بر اساس تعداد نودهای تولید شده در حین جستجو و پیچیدگی مکانی بر اساس ماکزیمم تعداد نودهای ذخیره شده در حافظه اندازه گیری می‌شود.

انواع استراتژی‌های جستجو:

از نظر اینکه آیا از تعداد مراحل یا هزینه مسیر از حالت جاری به حالت هدف اطلاعاتی دارند یا نه، به دو دسته تقسیم می‌شوند:

I. جستجوهای آگاهانه :

این دسته از الگوریتم‌ها علاوه بر اطلاعات مسئله اطلاعات اضافی برای رسیدن به هدف در اختیار دارند که می‌توانند امید بخش تر بودن یک گره نسبت به گره دیگر را با توجه به اطلاعات اضافی تشخیص دهند که به این دسته جستجوها، جستجوهای اکتشافی نیز گفته می‌شود.

II. جستجوهای ناآگاهانه :

این دسته از الگوریتم‌های جستجو، هیچ اطلاعاتی غیر از تعریف مسئله در اختیار ندارند و فقط می‌توانند جانشین‌هایی را تولید و هدف را از غیر هدف تشخیص دهند که به آن جستجوهای کور کورانه نیز گفته می‌شود.

انواع جستجوهای ناآگاهانه عبارتند از:

- ❖ جستجوی اول-سطح
- ❖ جستجوی هزینه یکنواخت
- ❖ جستجوی اول-عمق
- ❖ جستجوی عمقی محدود شده
- ❖ جستجوی عمقی تکرار شونده
- ❖ جستجوی دو طرفه

function TREE-SEARCH(*problem*) **returns** a solution, or failure
 initialize the frontier using the initial state of *problem*
 loop do
 if the frontier is empty **then return** failure
 choose a leaf node and remove it from the frontier
 if the node contains a goal state **then return** the corresponding solution
 expand the chosen node, adding the resulting nodes to the frontier

function GRAPH-SEARCH(*problem*) **returns** a solution, or failure
 initialize the frontier using the initial state of *problem*
 initialize the explored set to be empty
 loop do
 if the frontier is empty **then return** failure
 choose a leaf node and remove it from the frontier
 if the node contains a goal state **then return** the corresponding solution
 add the node to the explored set
 expand the chosen node, adding the resulting nodes to the frontier
 only if not in the frontier or explored set

Figure 3.7 An informal description of the general tree-search and graph-search algorithms. The parts of GRAPH-SEARCH marked in bold italic are the additions needed to handle repeated states.

جستجوهای آگاهانه

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes ^a	Yes ^{a,b}	No	No	Yes ^a	Yes ^{a,d}
Time	$O(b^d)$	$O(b^{1+\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes ^c	Yes	No	No	Yes ^c	Yes ^{c,d}

Figure 3.21 Evaluation of tree-search strategies. b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit. Superscript caveats are as follows: ^a complete if b is finite; ^b complete if step costs $\geq \epsilon$ for positive ϵ ; ^c optimal if step costs are all identical; ^d if both directions use breadth-first search.

جستجوهای ناآگاهانه

جستجوی آگاهانه

- علاوه بر تعریف مساله، دانش و اطلاعاتی درباره انتخاب راه حل وجود دارد.
- نگرش کلی جستجوهای آگاهانه انتخاب اولین-بهترین (Best-first) براساس تابع ارزیابی ($f(n)$) است.
- میتوان گره‌های باز (بسط پیدا نکرده) را در صف اولویت قرار داد.
- تابع $f(n)$ فاصله تا گره هدف را نشان می‌دهد.
- نکته کلیدی جستجوهای آگاهانه استفاده از تابع هیوریستیک می‌باشد.
- دانش اضافه شده به مساله، همین تابع هیوریستیک می‌باشد، که فاصله گره n تا هدف را تخمین می‌زند.

جستجوی اول بهترین

□ ایده.

□ هر گره‌ای که تولید می‌شود را **ارزیابی** کن. [تابع ارزیابی]

□ هر بار **بهترین** گره گسترش نیافته را انتخاب و گسترش بده.

□ **پیاده‌سازی**. با استفاده از یک **صف اولویت** که در آن گره‌ها بر اساس میزان مطلوب بودنشان مرتب شده‌اند.

تابع ارزیابی

$$f(n) = h(n)$$

$$f(n) = g(n) + h(n)$$

□ انواع خاص از جستجوی اول بهترین.

□ جستجوی حریصانه

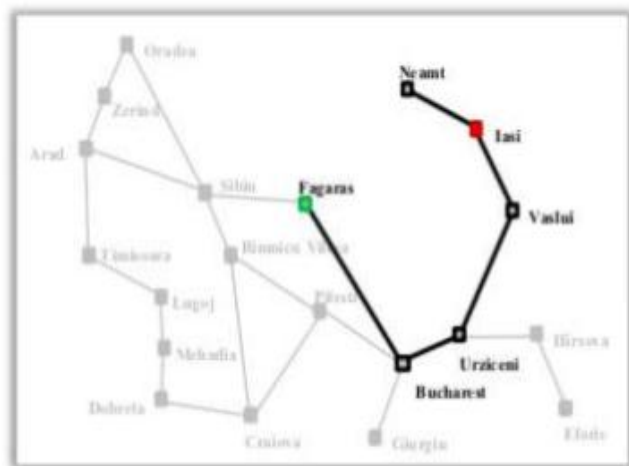
□ جستجوی A^*

جستجوی اولین-بهترین حریصانه
(Greedy best-first Search)

این جستجو با استفاده از تابع هیوریستیک گره‌ها را برای بسط انتخاب می کند. $f(n)=h(n)$

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

ارزیابی جستجوی حریصانه



□ کامل؟ خیر [ممکن است در یک حلقه‌ی بی نهایت گیر کند]

□ مثال: جستجو از یاش به فگره

□ بهینه؟ خیر [مثال رومانی]

$$b^1 + b^2 + b^3 + \dots + b^m \in O(b^m)$$

□ پیچیدگی زمانی؟ نمایی [مانند جستجوی عمقی]

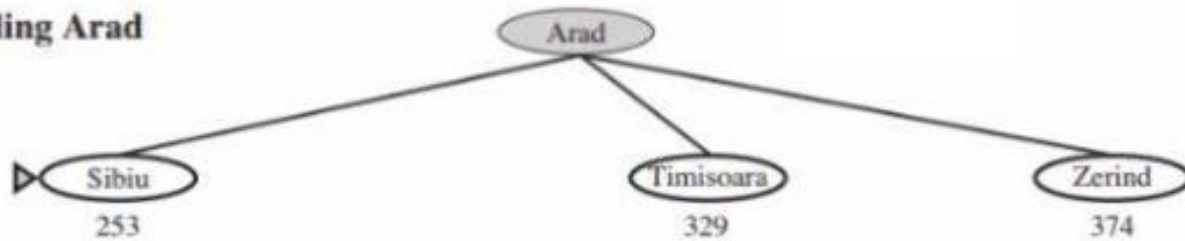
$$O(b^m)$$

□ پیچیدگی حافظه؟ نمایی

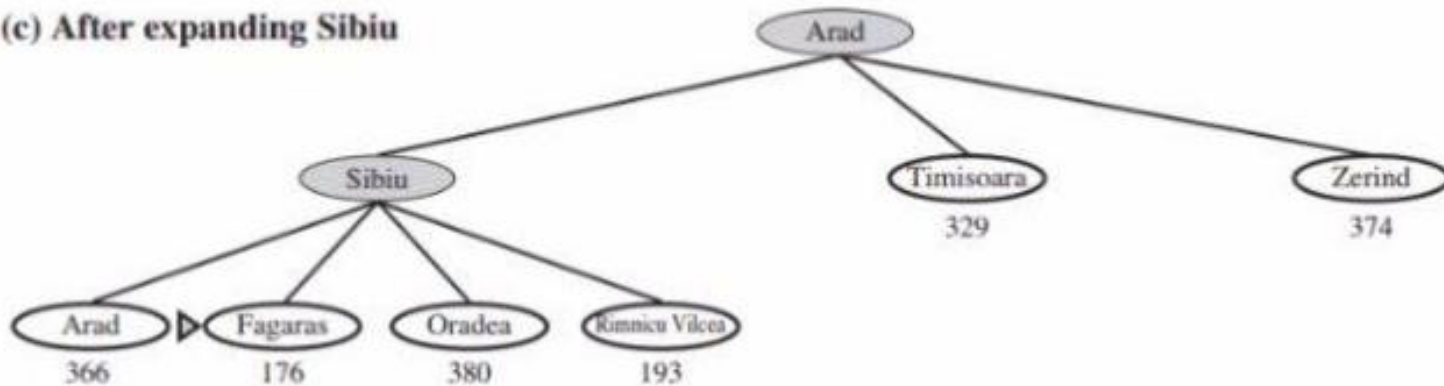
(a) The initial state



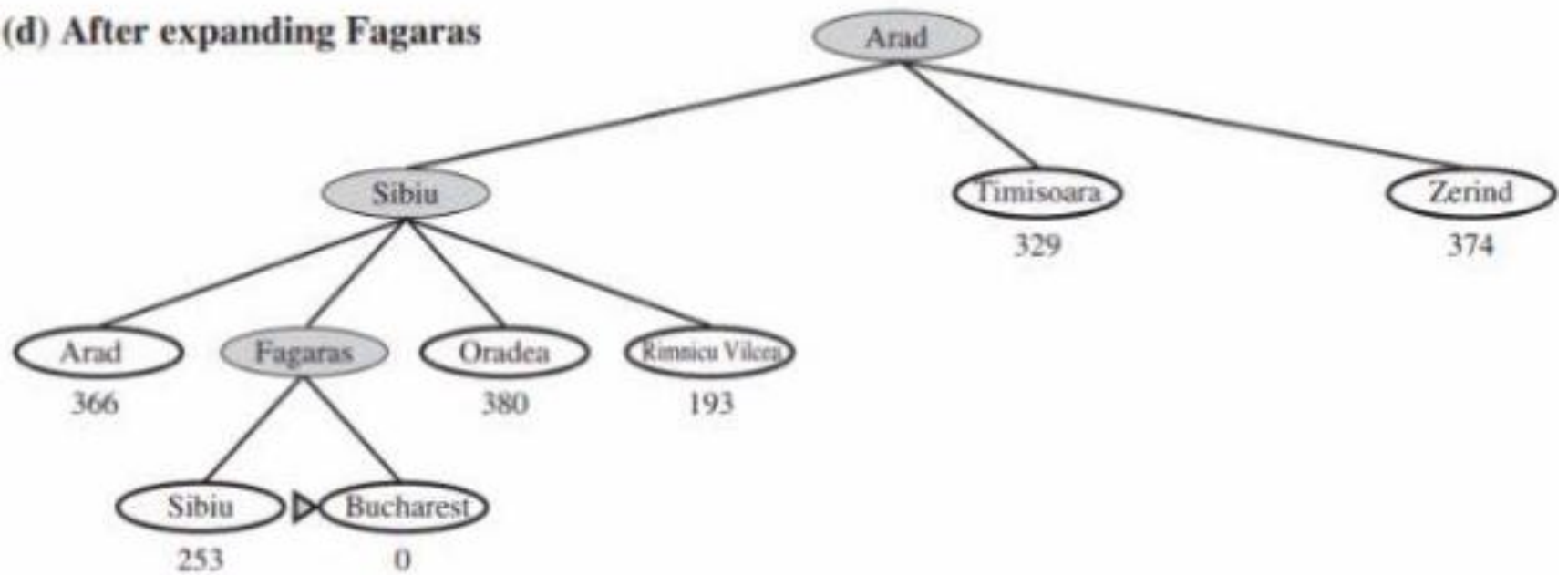
(b) After expanding Arad

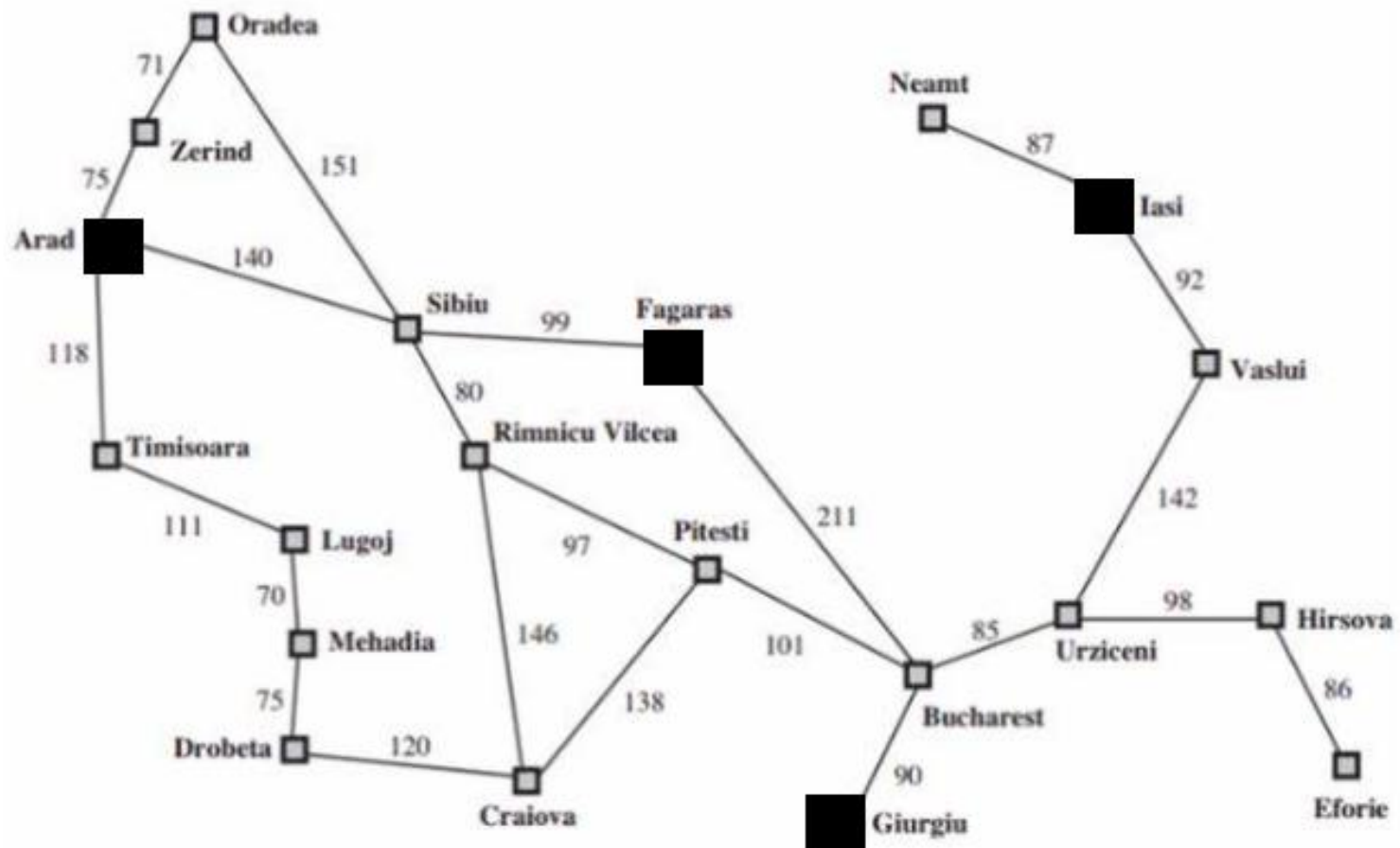


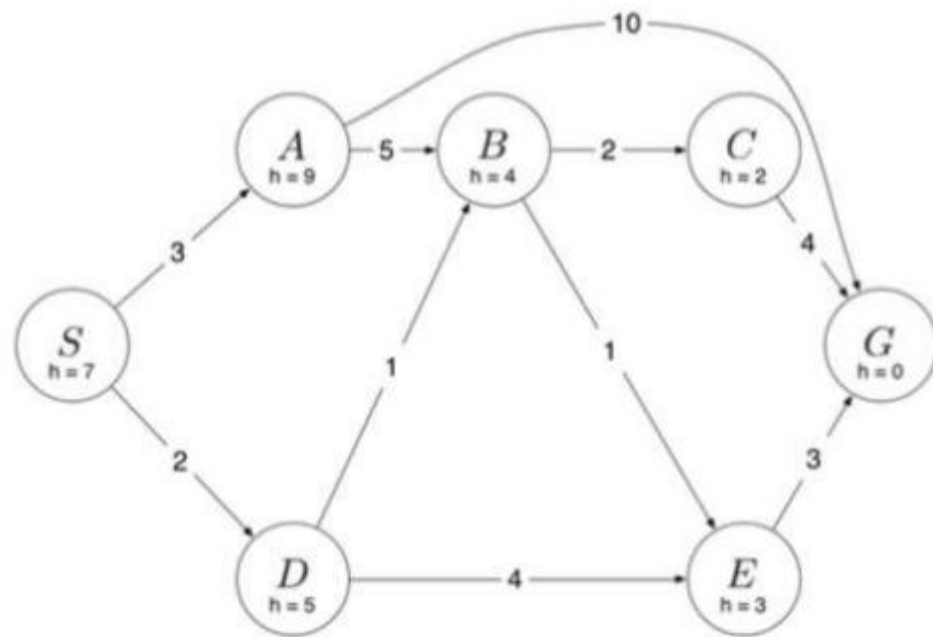
(c) After expanding Sibiu



(d) After expanding Fagaras







جستجوی A^*

معروفترین جستجوی، اولین بهترین می باشد.

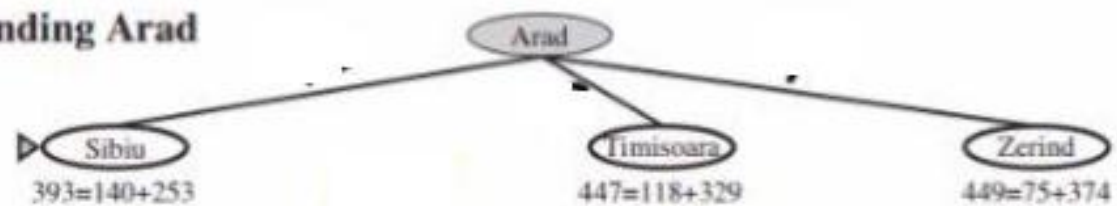
در این روش برای ارزیابی گره ها هم از تابع هیوریستیک و هم از فاصله گره شروع از گره کنونی استفاده می کند.

در صورتیکه فضای جستجو درخت باشد، و تابع هیوریستیک قابل قبول (admissible) باشد، الگوریتم A^* بهینه است.

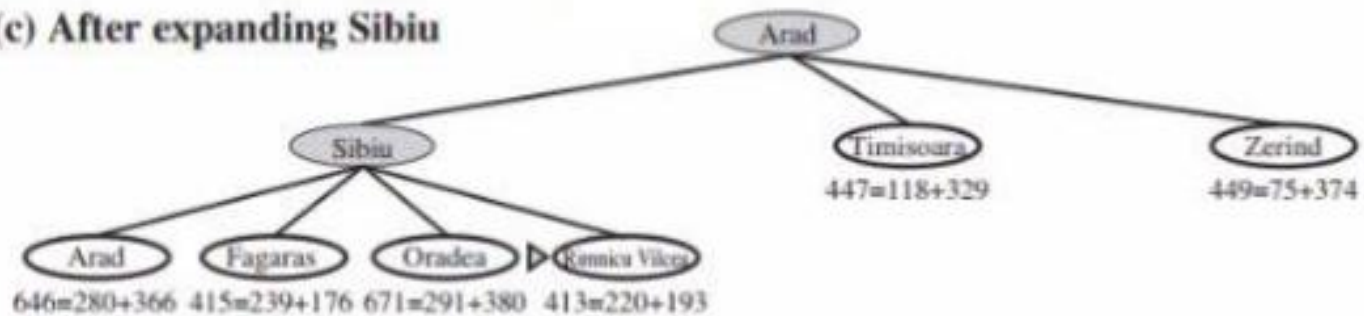
(a) The initial state



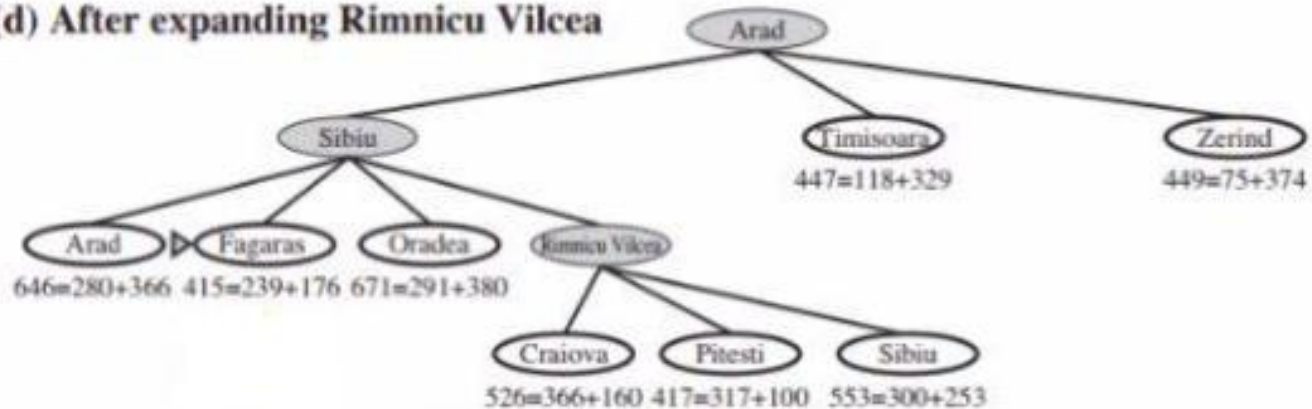
(b) After expanding Arad



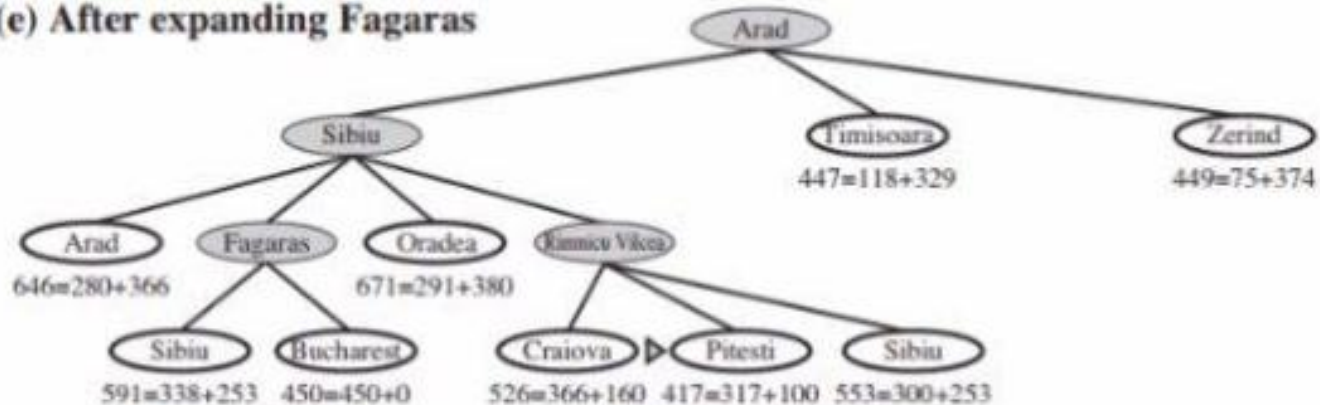
(c) After expanding Sibiu



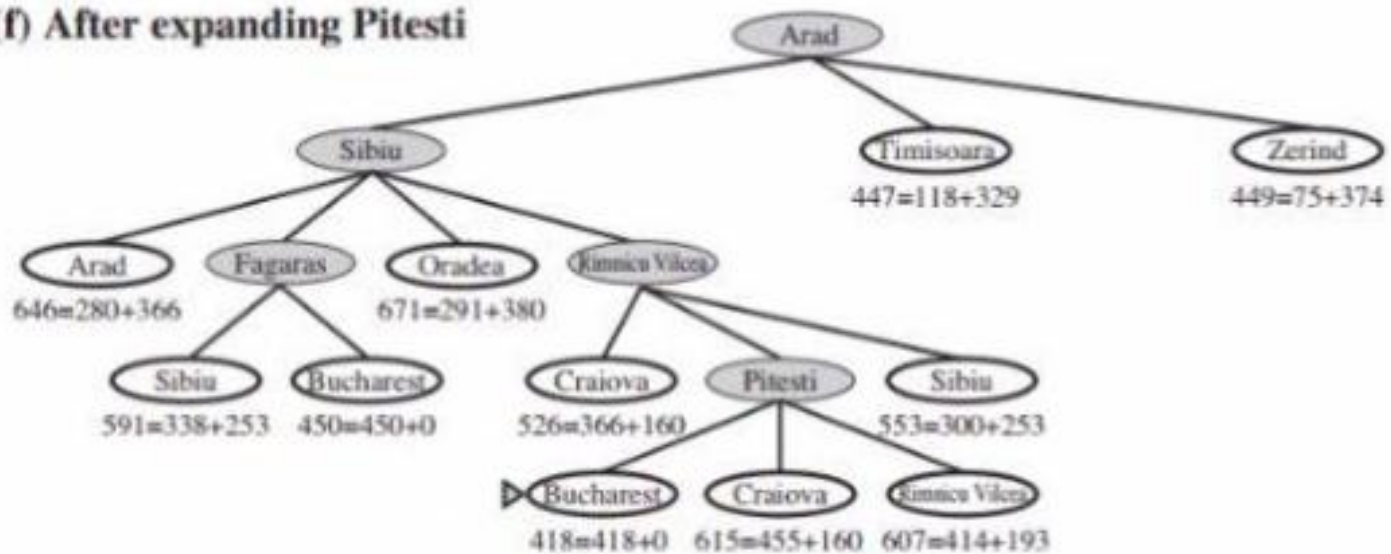
(d) After expanding Rimnicu Vilcea



(e) After expanding Fagaras

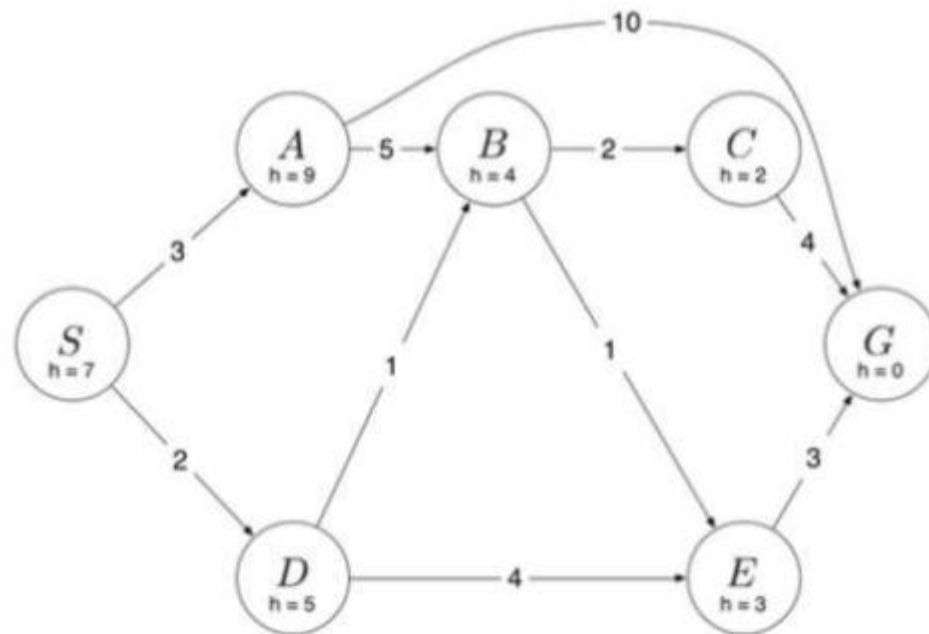


(f) After expanding Pitesti



الگوریتم A^* در جستجوی گرافی ممکن است مسیر بهینه را به دلیل حذف یک حالت تکراری کنار بگذارد.

لذا برای بهینه بودن الگوریتم A^* در جستجوی گرافی، باید خاصیت سازگاری (یکنوایی) در تابع هیوریستیک وجود داشته باشد.



آیا جستجوی A^* بهینه است؟

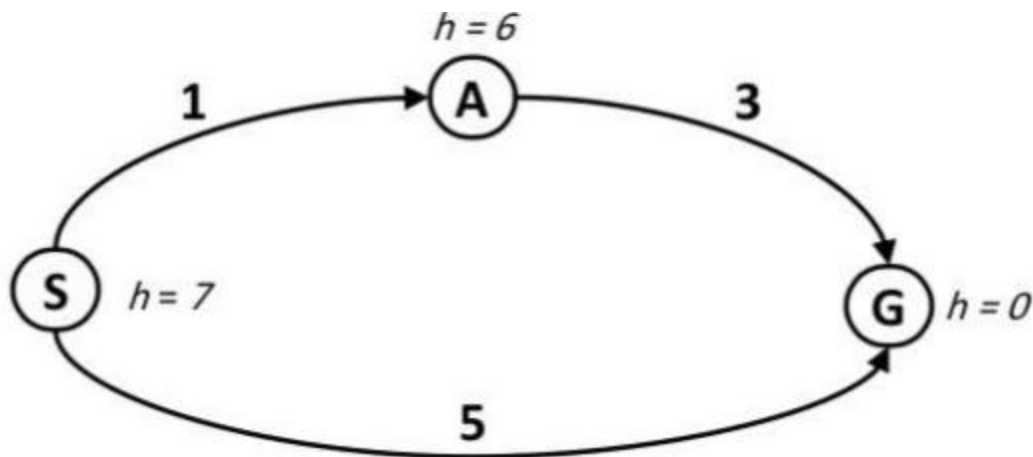
□ بله، به شرطی که تابع هیورستیک قابل قبول باشد.

هیورستیک قابل قبول. تابع هیورستیک $h(n)$ قابل قبول است اگر به ازای هر گره مانند n همواره داشته باشیم:

$$0 \leq h(n) \leq h^*(n)$$

به طوری که $h^*(n)$ بیانگر هزینه واقعی کوتاه‌ترین مسیر از n تا هدف است.

مثال. هیورستیک فاصله مستقیم در مسایل مسیریابی.

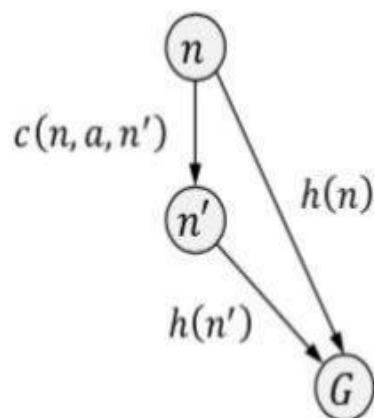


آیا **A*** در گراف بالا مسیر بهینه را برمی گرداند؟ ایراد کار در کجاست؟

هزینه‌ی **واقعی** مسیر بد $>$ هزینه‌ی **تخمینی** مسیر خوب
بنابراین تخمین‌ها باید از مقدار واقعی کمتر باشند.

هیوریستیک سازگار. یک هیوریستیک سازگار است اگر:

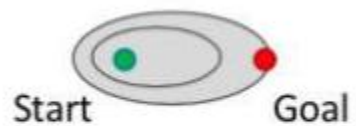
$$h(n) \leq c(n, a, n') + h(n')$$



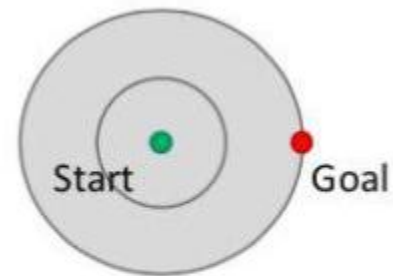
یکنوایی. اگر h سازگار باشد، آنگاه f در طول هر مسیری غیرکاهشی است.

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n, a, n') \\ &\geq g(n) + h(n) \\ &= f(n) \end{aligned}$$

قضیه. اگر h سازگار باشد، آنگاه A^* در جستجوی گراف بهینه است.

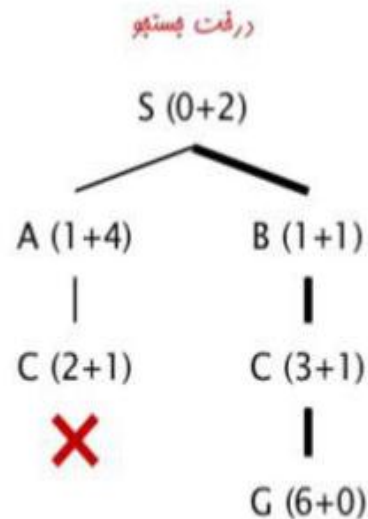
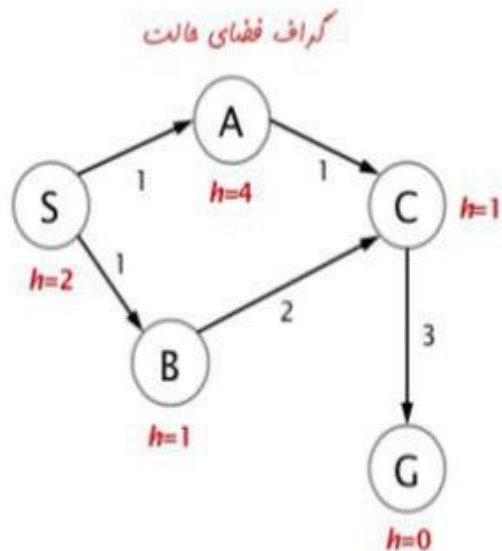


جستجوی A^* عمدتاً گره‌ها را در جهت هدف گسترش می‌دهد، اما برای تضمین بهینگی گاهی اوقات نیز به سمت اطراف حرکت می‌کند.

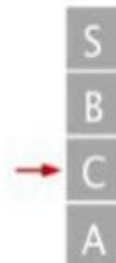


جستجوی هزینه یکنواخت بدون توجه به هدف، در همه جهات‌ها به طور یکسان جستجو می‌کند.

□ مثال. یافتن مسیر از S به G.

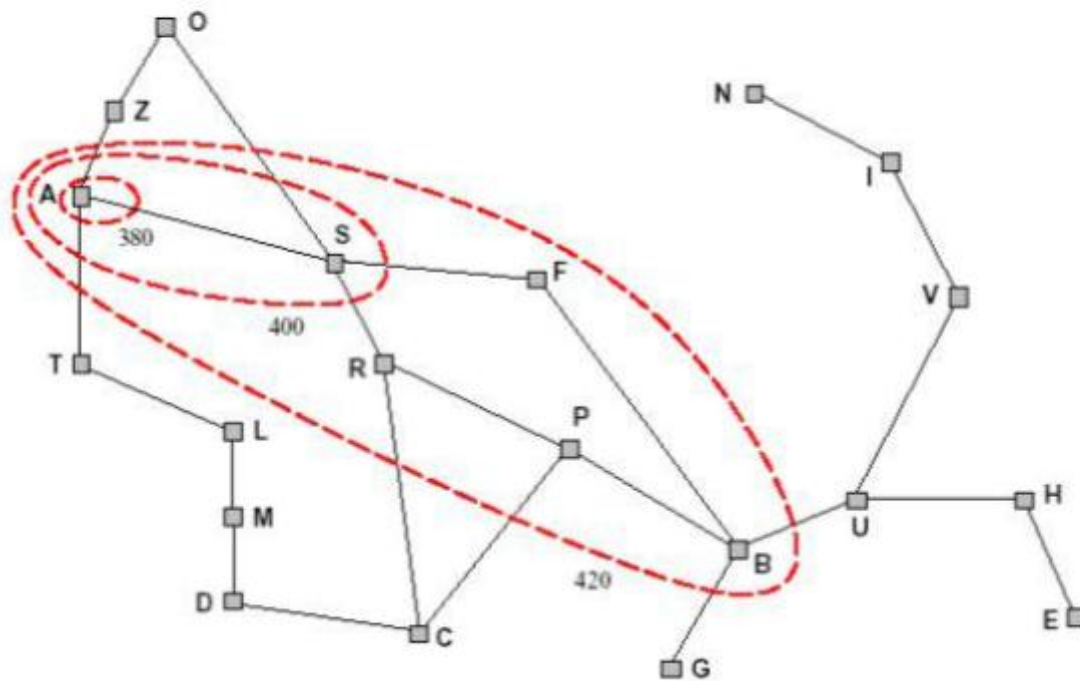


فالت‌های
گسترش یافته



□ توجه. در این مثال اگرچه تابع هیوریستیک قابل قبول است، اما A* قادر به یافتن راه حل بهینه نیست.

A^* گره‌ها را به ترتیب صعودی مقادیر f گسترش می‌دهد:
 A^* به ترتیب کانتورهای f را از کوچک به بزرگ اضافه می‌کند.



جستجوی درختی.

- جستجوی A^* بهینه است اگر هیوریستیک **قابل قبول** باشد.
- جستجوی هزینه یکنواخت یک حالت خاص است ($h = 0$)

جستجوی گراف‌ی.

- جستجوی A^* بهینه است اگر هیوریستیک **سازگار** باشد.
- جستجوی هزینه یکنواخت بهینه است ($h = 0$ سازگار است)

هر تابع هیوریستیک سازگار، قابل قبول نیز هست.

- به طور کلی، اغلب توابع هیوریستیک قابل قبول طبیعی سازگار نیز هستند، به ویژه اگر از مسائل راحت شده به دست آمده باشند.

ارزیابی A^*

کامل. بله، مگر آن که تعداد نامحدودی گره با $f(n) \leq f(G)$ وجود داشته باشد.

□ گره‌ای با فاکتور انشعاب نامحدود وجود داشته باشد.

□ مسیری با هزینه‌ی محدود اما تعداد گره‌های نامحدود وجود داشته باشد. (عملگری با هزینه منفی یا صفر وجود داشته باشد)

پیچیدگی زمانی. نمایی، برحسب خطای نسبی هیوریستیک ضرب در هزینه‌ی راه‌حل.

□ مگر آن که نرخ رشد خطای تابع هیوریستیک بیشتر از لگاریتم هزینه‌ی مسیر واقعی نباشد.

$$|h(n) - h^*(n)| \leq O(\log h^*(n))$$

پیچیدگی حافظه. نمایی، زیرا تمام گره‌های تولید شده را در حافظه نگه می‌دارد.

بهینه. بله، نمی‌تواند کانتور f_{i+1} را گسترش دهد مگر آن که f_i تمام شده باشد.

□ تمامی گره‌ها با $f(n) < f^*$ گسترش داده می‌شوند.

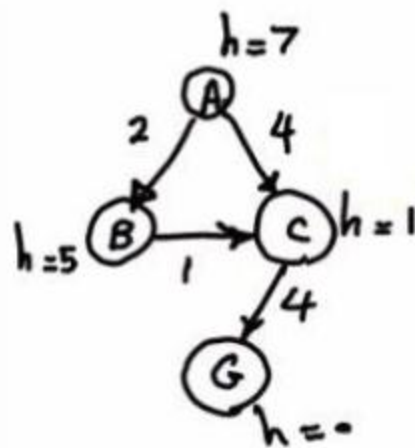
□ برخی گره‌ها با $f(n) = f^*$ گسترش داده می‌شوند.

□ هیچ گره‌ای با $f(n) > f^*$ گسترش داده نمی‌شود.

A^* دارای کارایی بهینه است.

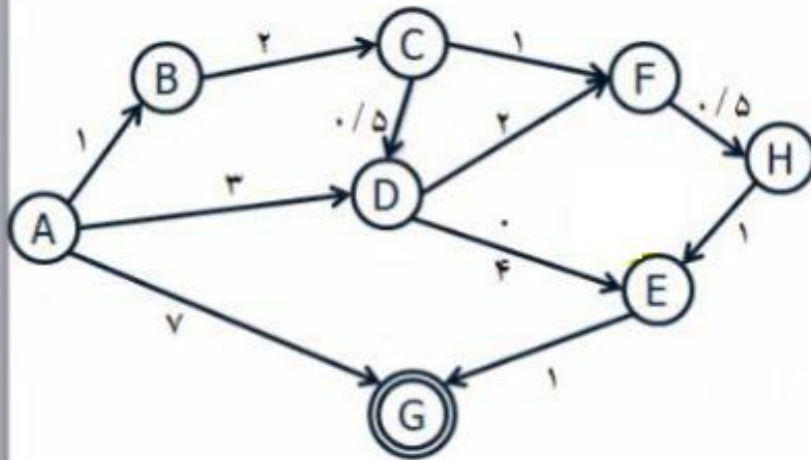
□ یعنی در بین تمام استراتژی‌های بهینه، کمترین تعداد گره‌های ممکن را تولید می‌کند.

بررسی بهینگی A^*



مثال

باتوجه به گراف جستجوی زیر، با شروع از حالت A، و توابع هیوریستیک داده شده با استفاده از الگوریتم A^* مسیر بهینه را بیابید.

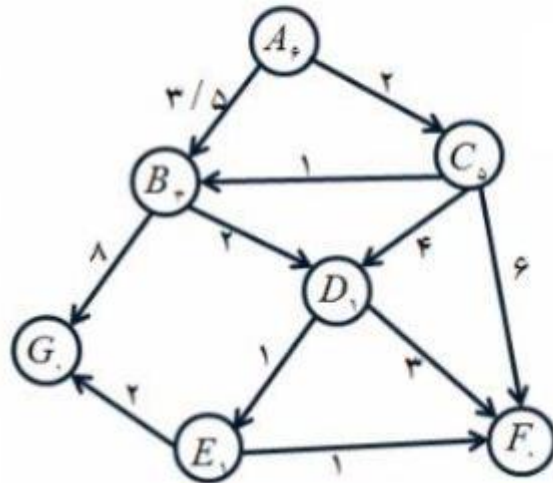


گره ها	h_1	h_2	h_3
A	۶	۶/۵	۵
B	۴	۵	۵/۵
C	۲	۳	۳/۵
D	۵	۳	۴
E	۱	۱	۰/۵
F	۳/۵	۲	۲
H	۲	۲	۱/۵
G	۰	۰	۰

جستجوی عرضی، حالت خاصی از جستجوی هزینه یکنواخت است.
جستجوهای عرضی، عمقی و جستجوی یکنواخت، حالت‌های خاصی از جستجوی اول-بهترین هستند
جستجوی هزینه یکنواخت، حالت خاصی از جستجوی A^* است.

مثال

گراف مقابل را در نظر بگیرید. با استفاده از الگوریتم A^* چه مسیری را برگشت می دهد؟



نکاتی کلی الگوریتم A^*

- در الگوریتم A^* ، در هر مسیر تابع f ، غیر کاهشی است
- الگوریتم A^* ، تمام گره‌ها با $f(n) < C^*$ را گسترش می‌دهد.
- الگوریتم A^* ، ممکن است قبل از انتخاب گره هدف، تعدادی نود با $f(n) = C^*$ را گسترش دهد.
- اولین راه حل پیدا شده در الگوریتم A^* ، بهینه است.
- الگوریتم A^* ، هیچ گره‌ای با $f(n) > C^*$ را گسترش نمی‌دهد.
- هیچ الگوریتم بهینه‌ای نمی‌تواند تضمین کند، که تعداد گره کمتری نسبت به A^* ، بسط دهد.

الگوریتم IDA^*

- برای کاهش حافظه مورد نیاز در الگوریتم A^* ، میتوان مشابه روش جستجوی عمقی تکرار شونده عمل کرد.
- در این الگوریتم به جای عمق، محدودیت روی میزان $f(n)$ خواهد بود.
- مقدار اولیه محدودیت در این الگوریتم همان f ریشه است.
- در هر تکرار گره هایی که هزینه آنها از f -limit کمتر باشد، بسط می یابند.
- در صورت، پیدانشدن هدف، در تکرار i ام، آنگاه در تکرار $i+1$ ام، کمترین مقدار f گره های بسط پیدا نکرده بعنوان f -limit استفاده می شود. هدف زمانی پیدا می شود که $f\text{-limit} = C^*$.

الگوریتم IDA^* کامل و بهینه است.

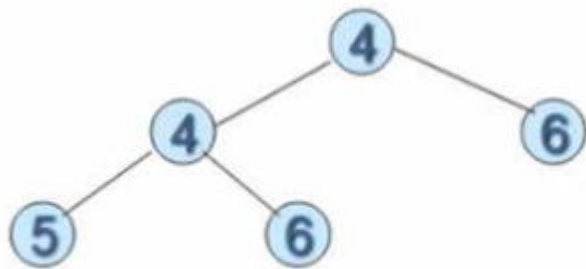
پیچیدگی مکانی آن مانند جستجوی عمقی، خطی است.

پیچیدگی زمانی این الگوریتم، به تعداد تغییرات f در طول مسیر پیدانمودن حل بهینه وابسته است.

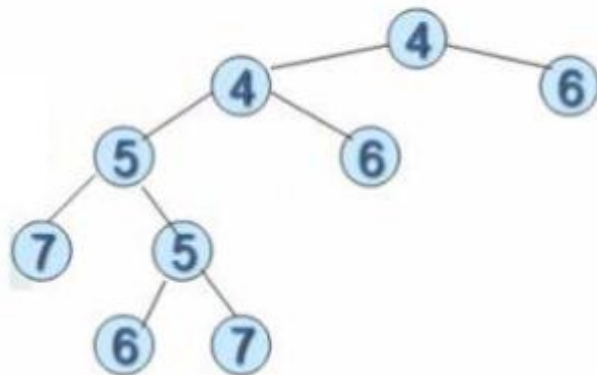
در مسایلی، که مقدار f گره‌ها، با هم فرق دارد، در هر تکرار فقط یک نود بیشتر از تکرار قبلی بسط پیدا می کند.

اگر A^* ، n گره را بسط دهد، آنگاه IDA^* ، در بدترین حالت n^2 گره را بسط می دهد.

F-Limit = 4



F-Limit = 5



الگوریتم اول-بهترین بازگشتی (RBFS)

این جستجو با تقلید از جستجوی اولین-بهترین عمل می کند، اما پیچیدگی مکانی آن خطی است.

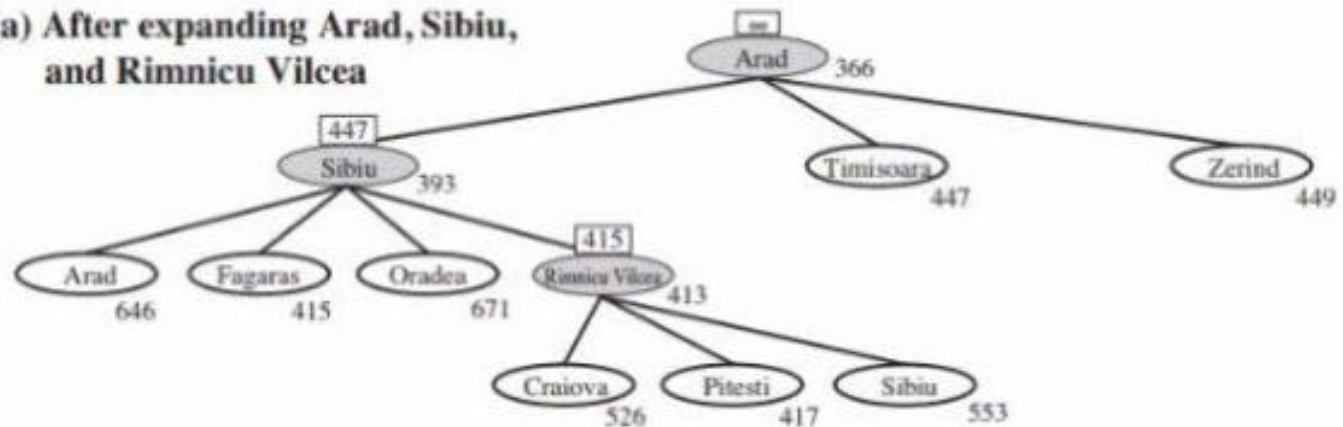
همانند جستجوی عمقی، بازگشتی عمل می کند، با این تفاوت که تا آخرین گره از مسیر فعلی را نخواهد رفت در واقع مقدار f بهترین مسیر جایگزین از طریق اجداد گره فعلی را نگه می دارد. و در صورتیکه f گره فعلی از این مقدار جایگزین بیشتر باشد، الگوریتم به عقب برمی گردد.

در برگشت به عقب الگوریتم، در طول مسیر، بهترین مقدار f فرزندان هر گره را بجای f آن گره قرار می دهد.

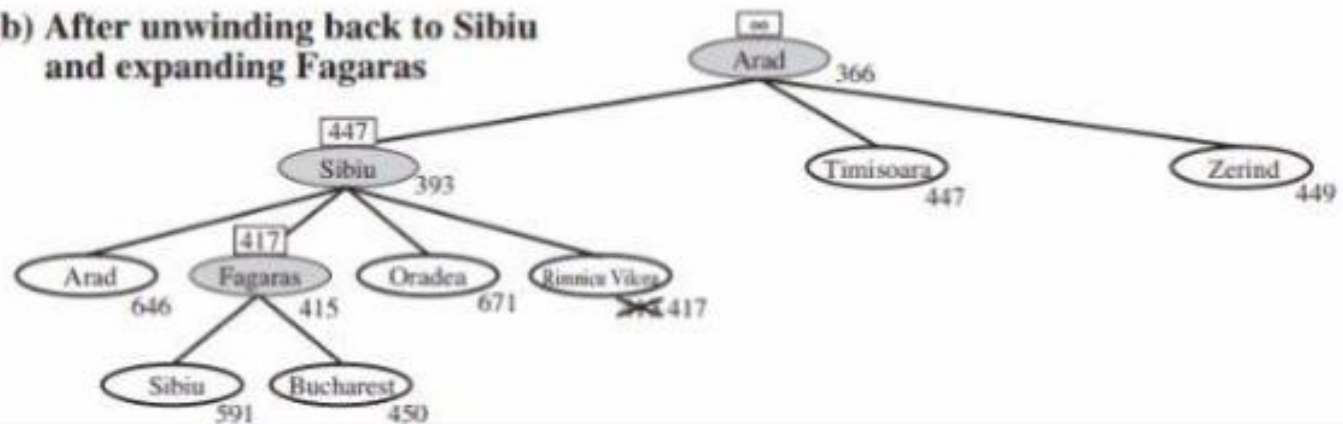
در صورت قابل قبول بودن تابع اکتشافی، RBFS همانند A^* بهینه است.

کارایی بهتری نسبت به IDA^* دارد. اما همچنان مشکل بسط تکراری بعضی از گره ها را دارد.

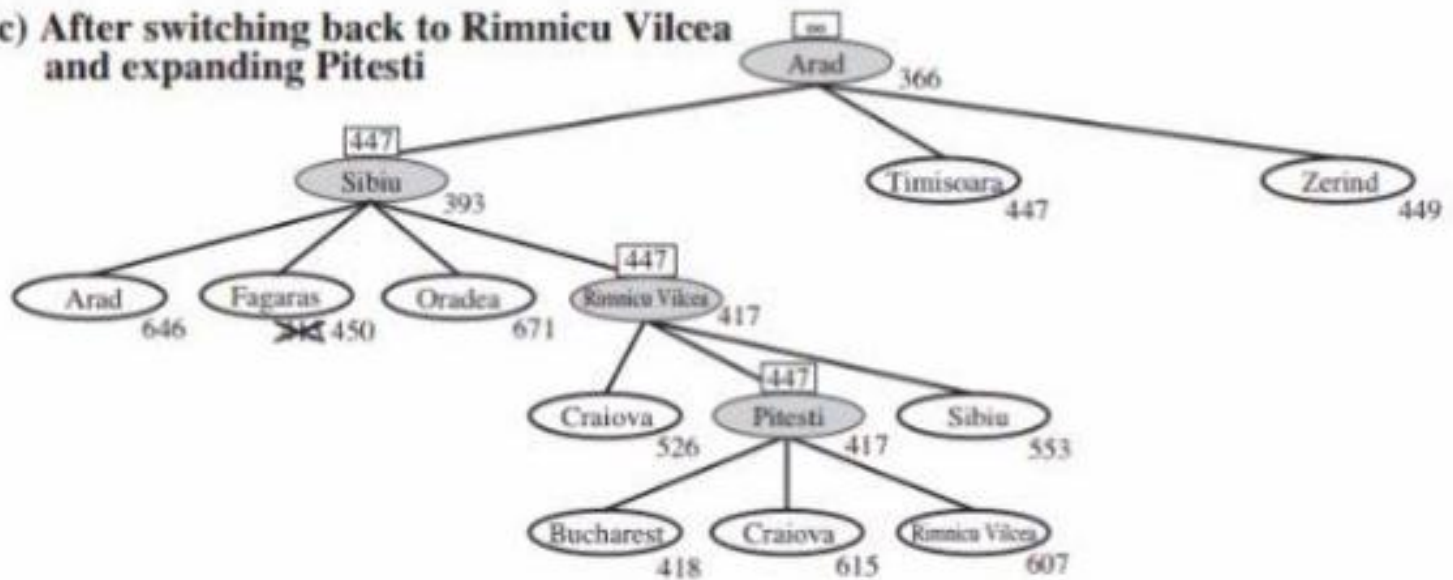
(a) After expanding Arad, Sibiu, and Rimnicu Vilcea



(b) After unwinding back to Sibiu and expanding Fagaras



(c) After switching back to Rimnicu Vilcea and expanding Pitesti



الگوریتم A^* Memory Bounded

الگوریتم های IDA^* و RBFS از میزان حافظه دسترس بخوبی نمی توانند استفاده کنند. لذا بهتر است الگوریتمی از الگوریتم های MA^* استفاده کنیم.

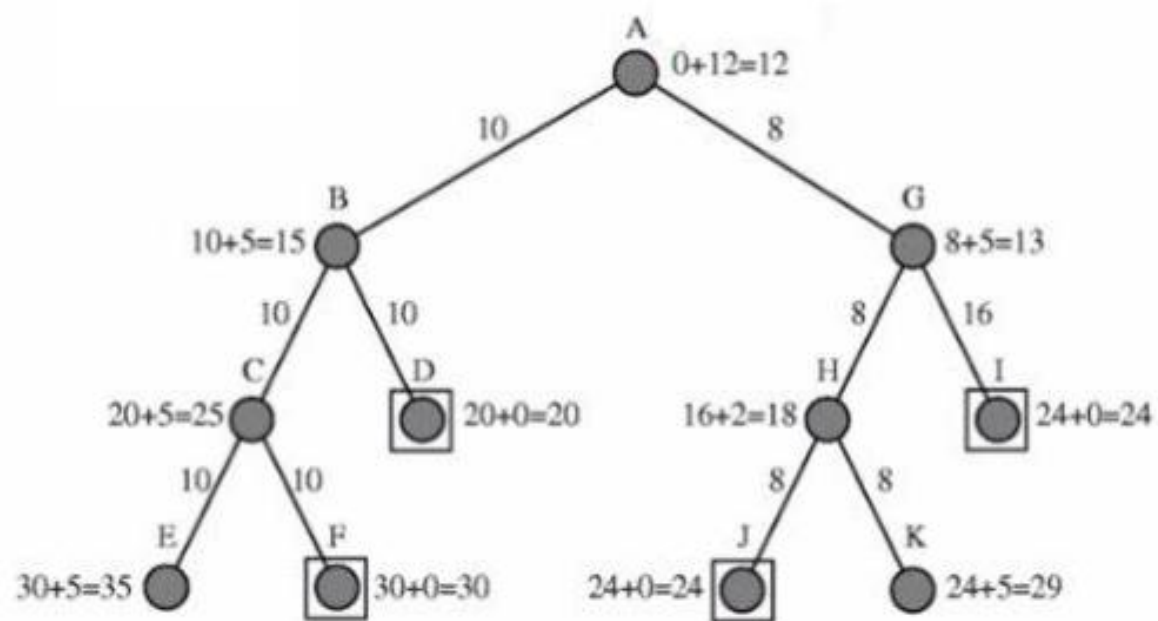
الگوریتم SMA^* ، مانند A^* ، بهترین برگ را بسط می دهد تا اینکه حافظه مورد دسترس آن پر گردد.

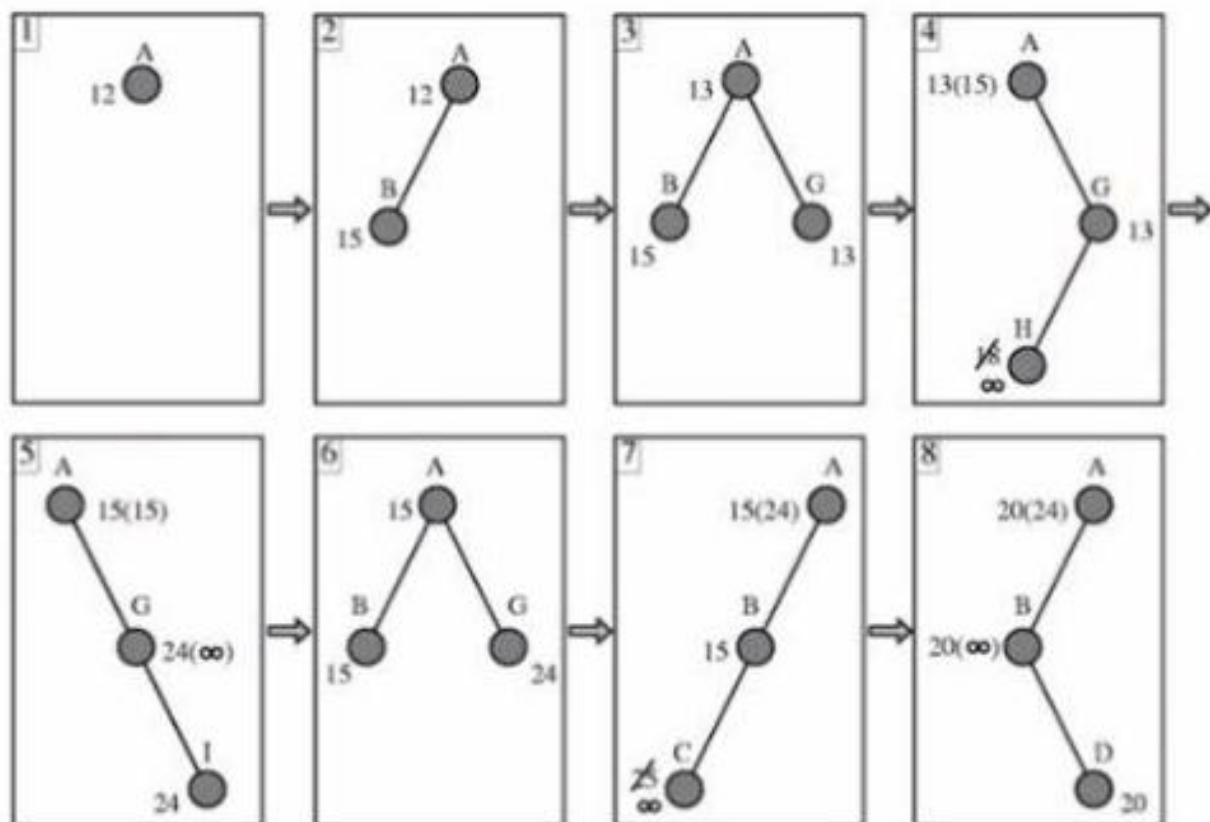
در صورتیکه، حافظه پر شود، و هدف پیدا نشده باشد، آنگاه بدترین گره برگ از حافظه حذف شده و مثل RBFS f گره حذف شده، به پدر آن داده می شود.

در صورتیکه الگوریتم، فقط یک گره برای بسط و حذف داشته باشد، در صورتیکه این گره برگ هدف نباشد، آنگاه الگوریتم در هر صورت این گره را حذف و مقدار f آن را بی نهایت قرار می دهد.

SMA^* در صورتی کامل است که ارتفاع کم عمقترین هدف از میزان حافظه در دسترس بیشتر نباشد.

این الگوریتم در صورتیکه حافظه مورد نیاز برای نگه‌داری مسیر هدف بهینه در دسترس باشد، بهینه می‌باشد.





مثال

با استفاده از الگوریتم SMA* با حافظه ۳ کدام مسیر برگشت داده می شود؟

